Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

Nachname/*Last name* Vorname/*First name* Matrikelnr./*Matriculation no*

Scheinklausur
02. 03. 2015

- Bitte tragen Sie zuerst auf dem Deckblatt Ihren Namen, Ihren Vornamen und Ihre Matrikelnummer ein. Tragen Sie dann auf den anderen Blättern (auch auf dem Konzeptblatt) Ihre Matrikelnummer ein.
  Please fill in your last name, your first name, and your matriculation number on this page and fill in your matriculation number on all other pages (including the draft page).
- Die Prüfung besteht aus 21 Blättern: 1 Deckblatt, 17 Aufgabenblättern mit insgesamt 5 Aufgaben und 3 Blätter Man-Pages.
  The examination consists of 21 pages: 1 cover sheet, 17 sheets containing 5 assignments, and 3 sheets for man pages.
- Es sind keinerlei Hilfsmittel erlaubt!
  No additional material is allowed.
- Die Prüfung gilt als nicht bestanden, wenn Sie versuchen, aktiv oder passiv zu betrügen.
  You fail the examination if you try to cheat actively or passively.
- Wenn Sie zusätzliches Konzeptpapier benötigen, verständigen Sie bitte die Klausuraufsicht.
  If you need additional draft paper, please notify one of the supervisors.
- Bitte machen Sie eindeutig klar, was Ihre endgültige Lösung zu den jeweiligen Teilaufgaben ist. Teilaufgaben mit widersprüchlichen Lösungen werden mit 0 Punkten bewertet.
  Make sure to clearly mark your final solution to each question. Questions with multiple, contradicting answers are void (0 points).
- Programmieraufgaben sind gemäß der Vorlesung in C zu lösen.
  Programming assignments have to be solved in C.

Die folgende Tabelle wird von uns ausgefüllt! *The following table is completed by us!*

Aufgabe	1	2	3	4	5	Total
Max. Punkte	12	12	12	12	12	60
Erreichte Punkte						
Schein (Bonus)						

Aufgabe 1: C Grundlagen

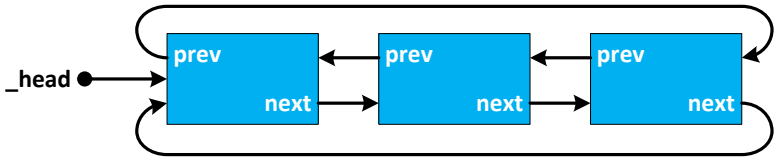
Assignment 1: C Basics

In der vorliegenden zirkulären, doppelt-verketteten Liste werden Elemente durch `entry` und die Zeiger auf das vorherige und nächste Element durch `prev` und `next` repräsentiert. Das letzte Element der Liste zeigt immer zurück auf das erste und umgekehrt. Der Kopf der Liste (`head`) zeigt auf das erste Element oder ist `NULL`, falls die Liste leer ist.

In the given circular, doubly-linked list, entries are represented by `entry` and the pointers to the previous and next entries by `prev` and `next`, respectively. The last entry always points back to the first one and vice versa. The list head (`head`) points to the first entry or is `NULL` if the list is empty.

```
typedef struct _entry {
    struct _entry *prev; // Pointer to previous entry in list
    struct _entry *next; // Pointer to next entry in list
    ...                  // Some data
} entry;

entry *_head = NULL;     // Pointer to first entry in list
```



a) Schreiben Sie eine Funktion, die ein neues Listenelement auf dem Heap alloziert. Bei Erfolg soll die Funktion einen Zeiger auf das Element zurückgeben, ansonsten `NULL`. **1 pt**

Write a function that allocates a new list entry on the heap. The function should return a pointer to the entry on success, `NULL` otherwise.

```
entry* allocateEntry(void) {
.....
.....
.....
.....
}
```

b) Schreiben Sie eine Funktion, die den Speicher eines bestehendes Listenelementes freigibt. Gehen Sie davon aus, dass das Element gültig (`e != NULL`) und nicht mehr Teil der Liste ist. **1 pt**

Write a function that frees the memory of an existing entry. Assume the entry to be valid (`e != NULL`) and not part of the list.

```
void freeEntry(entry *e) {
.....
.....
}
```

c) Schreiben Sie eine Funktion, die das Element `e` am Anfang der Liste einfügt. Gehen Sie davon aus, dass `e` auf ein gültiges Element zeigt (`e != NULL`). **3 pt**

Write a function that inserts the entry `e` at the head of the list. Assume `e` to point to a valid entry (`e != NULL`).

```
void insertEntry(entry *e) {
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
}
```

d) Schreiben Sie eine Funktion, die das Element `e` aus der Liste entfernt. Gehen Sie davon aus, dass `e` auf ein gültiges Element der Liste zeigt. **3 pt**

Write a function that removes the entry `e` from the list. Assume `e` to point to a valid entry in the list.

```
void removeEntry(entry *e) {
.....
.....
.....
.....
.....
.....
.....
.....
.....
```

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

}

e) Welche der folgenden Aussagen sind korrekt, welche sind inkorrekt? **2 pt**
(falsches Kreuz: -0.5P, kein Kreuz: 0P, korrektes Kreuz: 0.5P)

Which of the following statements are correct, which are incorrect?
(incorrectly marked: -0.5P, not marked: 0P, correctly marked: 0.5P)

korrekt/ *correct*	inkorrekt/ *incorrect*	
☐	☐	Der Code einer Funktion wird zusammen mit ihren lokalen Variablen im Stack-Segment abgelegt. *The code of a function is stored together with its local variables in the stack segment.*
☐	☐	Initialisierte globale Variablen liegen im Daten-Segment. *Initialized global variables are placed in the data segment.*
☐	☐	Globale Konstanten liegen im BSS-Segment. *Global constants are placed in the bss segment.*
☐	☐	`malloc()` erweitert dynamisch das Stack-Segment. *`malloc()` dynamically extends the stack segment.*

f) Was ist der Wert der Ausdrücke nach Ausführung des Programmschnipsels? **2 pt**

What is the value of the expressions after execution of the given code snippet?

```
int a[] =
  { 0x00112233, 0x44556677,
    0x8899aabb, 0xccddeeff,
    0xbaadf00d, 0xdeadc0de };

char b  = a[2];
int *p0 = &a[0];
int *p1 = p0 + sizeof(uint32_t);
// <-- Evaluate expression here
```

Expression	Value
`a[5] && 1`	
`b`	
`*p0`	
`*p1`	

Total: 12.0pt

Aufgabe 2: Prozesse und Threads

Assignment 2: Processes and Threads

Credit Scheduling arbeitet wie folgt: Jeder Thread im System verfügt über eine bestimmte Menge Laufzeit-"Guthaben". Immer, wenn ein Thread ausgeführt wird, verbraucht er einen Teil seines Guthabens. Wenn ein Thread die CPU abgibt, wird die verbrauchte Zeit von seinem Guthaben abgezogen, und der Scheduler wählt einen anderen Thread mit positivem Guthaben aus. Das Guthaben aller Threads wird periodisch um einen definierten Betrag bis zu einem definierten Maximum aufgefüllt. Guthaben, das nach dem Auffüllen dieses Maximum übersteigt, verfällt.

In dieser Aufgabe sollen Sie einen einfachen Credit Scheduler für ein Multiprozessorsystem implementieren.

- Denken Sie daran, dass das System nebenläufig ist. d.h. es kann mehrere, parallele Aufrufe Ihrer Funktionen geben. Nutzen Sie Funktionen der *pthreads*-Bibliothek zur Synchronisation.
- Definieren Sie, falls nötig, zusätzliche globale Variablen oder Datenstrukturen. Gehen Sie davon aus, dass alle Variablen mit 0 initialisiert werden.
- Sie müssen die Laufzeiten der Threads selbst erfassen. Messen Sie das Guthaben und den Verbrauch der Threads in CPU-Zyklen. Verwenden Sie dazu die Funktion `clock()`.
- Nehmen Sie an, dass es eine feste Anzahl von Threads (`NUM_THREADS`) im System gibt, und dass Threads nie beendet werden.

Credit scheduling works as follows: Each thread in the system has a certain runtime-"credit". Whenever the thread runs, it consumes some of that credit. Whenever a thread yields the CPU, the scheduler selects another thread with positive credit to run. The credit of all threads is replenished periodically by adding a defined amount up to a defined maximum. If after replenishing, a thread has more credit than the maximum, any credit exceeding the maximum expires.

Your objective is to write a simple credit scheduler for a multiprocessor system.

- *Remember that your functions can be called multiple times concurrently. Use functions from the* phthreads*-library for synchronization.*
- *If necessary, define additional global variables or data structures. Assume that all variables are initialized to zero.*
- *You need to account each thread's runtime yourself. Measure each thread's credit and consumption in cpu cycles using the function* `clock()`*.*
- *Assume that there is a fixed number of threads in the system (*`NUM_THREADS`*) and that threads never terminate.*

a) Implementieren Sie die Funktion `void schedule(void)`. Diese Funktion wählt einen gerade **nicht laufenden** Thread mit **positivem Guthaben** aus, der als nächstes laufen soll. Falls kein entsprechender Thread existiert, starten Sie stattdessen den Idle-Thread mit der Thread ID `IDLE_THREAD`. Nehmen Sie an, dass eine Funktion `void restart_thread(int thread_id)` definiert ist, mit der Sie den ausgewählten Thread starten können. Achten Sie darauf, dass kein Thread mit positivem Guthaben verhungert!

Hinweise:

- `schedule()` muss nicht zwingend zurückkehren.
- `schedule()` wird nur von `yield()` aufgerufen.

6 pt

*Implement the function `void schedule(void)`. This function chooses a currently **not executing** thread with **positive credit** to be run next. If there is no such thread, launch the idle thread with the ID `IDLE_THREAD` instead. Assume that there is a function `void restart_thread(int thread_id)` which launches the chosen thread. Make sure that threads with positive credit cannot starve!*

Hints:

- *It is ok if `schedule()` does not return.*
- *`schedule()` is only called by `yield()`.*

b) Implementieren Sie die Funktion `void yield(int thread_id)`. Diese Funktion wird von Threads aufgerufen, um die CPU abzugeben. Gehen Sie davon aus, dass jeder Thread nach endlicher Zeit `yield()` aufruft und seine Thread ID korrekt angibt.

Hinweis: Die Threads müssen `yield()` nicht aufrufen, bevor ihr Guthaben verbraucht ist. Das Guthaben der Threads darf negativ sein. **3 pt**

Implement the function `void yield(int thread_id)`. This function is called by threads wanting to give up the CPU for the time being. Assume that each thread eventually calls `yield()` and passes its correct thread ID.

***Hint:** Threads do not necessarily call `yield()` before their entire credit is spent. Threads may have negative credit.*

c) Implementieren Sie die Funktion `void replenish(void)`. Diese Funktion erhöht das Guthaben aller Threads um den definierten Betrag `RP_CREDIT`. Das Maximum, bis zu dem das Guthaben aufgefüllt wird, wird durch `RP_MAX` vorgegeben. **3 pt**

Implement the function `void replenish(void)`. This function increases each thread's credit by the defined amount `RP_CREDIT`. The maximum credit a thread may have at any given time is defined by `RP_MAX`.

```
#include <time.h>
#include <pthread.h>

#define NUM_THREADS     42
#define IDLE_THREAD     -1
#define RP_CREDIT       10000
#define RP_MAX          100000

pthread_mutex_t mutex; // Assume that the mutex is properly initialized.

void restart_thread(int thread_id); // Does not return.
```

```
// Declare global variables here

void schedule(void) {
```

```
}

void yield(int thread_id) {

}

void replenish(void) {

}
```

Total: 12.0pt

Aufgabe 3: Koordination von Prozessen

Assignment 3: Process Coordination

Implementieren Sie Synchronisationsprimitive mit Hilfe von atomaren Instruktionen. Verwenden Sie die unten gegebenen Funktionen.

In this assignment you will implement synchronization based on atomic instructions. You may use any of the functions given below.

```
// atomically add i to a variable
// in memory, return the previous
// value of the variable
int atomic_add(int *var, int i) {
    /* ... */
}
```

```
// atomically swap x with a variable
// in memory, then return the previous
// value of the variable
int atomic_swap(int *var, int x) {
    /* ... */
}
```

a) Implementieren Sie ein einfaches Spinlock. Schreiben Sie hierzu zwei Funktionen `acquire` und `release` zum Holen und Freigeben eines Spinlocks. Setzen Sie voraus, dass jedes Lock als `int`-Variable repräsentiert und mit `0` initialisiert ist. Ihre Implementierung soll alle 1000 Iterationen *busy waiting* die CPU freiwillig abgeben (yield). **3 pt**

Implement a simple spinlock. Write two functions `acquire` and `release` that acquire and release a spinlock, respectively. Assume that each lock is represented by an `int` variable, which is initialized to `0`. Your implementation should voluntarily give up the CPU (yield) every 1000 iterations of busy waiting.

b) Begründen Sie, warum ein einfaches Spinlock keine Fairness garantiert. **1 pt**

Explain why a simple spinlock does not guarantee fairness.

c) Implementieren Sie eine Barrieren-Synchronisation als Funktion `void barrier()`. Die Funktion muss sich von mehreren Threads aufrufen lassen und darf erst zurückkehren, wenn alle beteiligten Threads sie betreten haben. Benutzen Sie eine der oben definierten atomaren Funktionen. Sie dürfen globale Variablen als Datenstruktur der Barriere einsetzen. Ihre Barriere braucht nur einmal zu funktionieren, Sie brauchen sie nicht zurückzusetzen. **4 pt**

Implement a barrier synchronization as a function `void barrier()`. That function must be safe to be called by multiple threads and must only return once all participating threads have entered. Base your code on one of the atomic functions defined above. You may introduce global variables as the barrier's data structures. Your barrier needs to work only once, no need to re-initialize it.

```
extern int N; // number of participating threads
```

d) Implementieren Sie eine Barrieren-Synchronisation wie in der vorherigen Teilaufgabe. Stellen Sie nun sicher, dass Ihre Implementierung mehrfach benutzt werden kann. Hinweise: Überlegen Sie sich, welcher Thread die Datenstruktur zurücksetzen soll. Wann ist ein geeigneter Zeitpunkt dafür? Ein Thread könnte erneut die Barriere betreten, bevor alle anderen Threads sie verlassen haben. **4 pt**

Implement a barrier as in the assignment above. However, make sure that your barrier implementation can be called several times in a row. Hints: Which thread should reset the barrier's data? When is a safe point in time to do that? A thread could enter the barrier again while other threads are still leaving.

```
extern int N; // number of participating threads
```

Total: 12.0pt

Aufgabe 4: Speicher

Assignment 4: Memory

Bei der Clock-Seitenersetzungsstrategie werden alle gültigen Seiten in einer zirkulären Liste gehalten. Jede Seite verfügt über ein Referenz-Bit (R-Bit), welches die MMU bei Zugriff auf 1 setzt. Der Zeiger der Uhr startet auf der ersten Seite. Bei Speicherdruck betrachtet der Algorithmus die aktuelle Seite nach folgender Regel:

- R-Bit 0 $\rightarrow$ Verwende Seite für Ersetzung, bewege Zeiger eine Seite weiter
- R-Bit 1 $\rightarrow$ Setze R-Bit auf 0 und suche weiter

In dieser Aufgabe sollen Sie den Algorithmus implementieren. Gehen Sie dabei von folgenden Rahmenbedingungen aus:

- Eine Seite ist über ihre Nummer (VPN) eindeutig definiert.
- Das System hat 1024 physische Seiten, die alle stets in Verwendung sind.
- Seiten werden nicht freigegeben, sondern nur durch den Algorithmus ersetzt.
- Eine physische Seite wird maximal von einer virtuellen Seite abgebildet.

The clock page replacement algorithm holds all valid pages in a circular list. Every page has a reference bit (r-bit), which the MMU sets to 1 on access. The clock hand starts at the first page. On memory pressure the algorithm examines the current page according to the following rule:

- *r-bit 0 $\rightarrow$ Use page as victim and advance hand*
- *r-bit 1 $\rightarrow$ Set r-bit to 0 and continue scanning*

In this question you will implement the clock algorithm. You can assume the following constraints:

- *Every page is uniquely identified by its number (vpn).*
- *The system has 1024 page frames, which are all permanently in use.*
- *Pages are not freed, but only replaced by the algorithm.*
- *A page frame is mapped at most by one page.*

a) Deklarieren Sie für Ihre Implementierung alle nötigen Variablen, um den Zustand der Uhr abzubilden. Setzen Sie passende Initialwerte. Sie müssen keine Initialisierung der verwendeten Seitenliste durchführen. **3 pt**

For your implementation, declare all necessary variables to represent the state of the clock. Set appropriate initial values. You do not have to initialize the clock's page list.

b) Implementieren Sie die Funktion `int replacePage(int new_vpn)`, die gemäß dem Clock-Algorithmus nach einer Seite sucht, die durch die neue Seite `new_vpn` ersetzt wird. Der Rückgabewert der Funktion ist die Nummer der zu ersetzenden Seite. Verwenden Sie die Funktionen `getRBit()` und `resetRBit()`, um auf das Referenz-Bit einer Seite zuzugreifen. **6 pt**

Implement the function `int replacePage(int new_vpn)`, which uses the clock algorithm to search for a page that should be replaced with the new page `new_vpn`. The function should return the number of the selected page for replacement. Use the functions `getRBit()` and `resetRBit()` to access a page's reference bit.

```
// Returns the status of the reference bit (1 set, 0 not set) for the
// page with the number supplied in vpn.
int getRBit(int vpn);

// Resets the reference bit of the supplied page to 0.
void resetRBit(int vpn);

#define NUM_FRAMES 1024 // Number of frames in the system

// Declare clock variables here
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................

// Returns the number of the page selected for replacement.
int replacePage(int new_vpn) {
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
}
```

c) Welche andere Seitenersetzungsstrategie wird durch den beschriebenen Algorithmus approximiert? **1 pt**

Which other page replacement policy does the given algorithm approximate?

d) Welche zusätzliche Information müsste der Clock-Algorithmus berücksichtigen, um geteilten Speicher (Shared Memory) zu unterstützen? **1 pt**

What additional information would the clock algorithm need to consider to support shared memory?

e) Bewerten Sie folgende Aussage: "Der Systemaufruf `malloc()` dient zur dynamischen Allozierung von Benutzerspeicher". Begründen Sie Ihre Aussage. **1 pt**

Evaluate the following statement: "The `malloc()` system call dynamically allocates user-space memory". Explain your answer.

Total: 12.0pt

Reserveplatz:/*Extra space:*

Aufgabe 5: Dateizugriff

Assignment 5: File Access

Eine gegebene Bibliothek abstrahiert Dateizugriffsoperationen. Es sind zwei Funktionen `isFileReadable()` und `isFileWriteable()` implementiert, die die Zugriffsrechte des aktuellen Benutzers auf eine Datei überprüfen.

A given library abstracts file access operations. Two functions `isFileReadable()` and `isFileWriteable()` are defined, which check the user's access permissions on a file.

```
// Returns 1, if the file can be read, 0 otherwise.
int isFileReadable(const char *filename){
        return (open(filename, O_RDONLY) >= 0) ? 1 : 0;
}
```

a) Welches Problem kann bei einer häufigen Verwendung dieser Funktion auftreten? **1 pt**

Which problem can occur, if this function is used frequently?

b) Implementieren Sie die Funktion `isFileWriteable()`, die überprüft ob für die gegebene Datei Lese- und Schreibrechte vorhanden sind. Vermeiden Sie das in Teilaufgabe a) identifizierte Problem. **1 pt**

Implement the function `isFileWriteable()`, which checks if the caller has read and write access to the given file. Avoid the problem identified in question a).

```
// Returns 1, if the file can be read and written, 0 otherwise.
int isFileWriteable(const char *filename) {
.....
.....
.....
.....
.....
.....
.....
.....
.....
}
```

c) Was stellen diese Funktionen auf einem multitaskingfähigem Betriebssystem nicht sicher? **1 pt**

What do these functions not ensure on a multi-tasking operating system?

Das Programm `tree` listet die Verzeichnisstruktur ausgehend von dem angegebenen Pfad hierarchisch auf. Das Programm zählt dabei wieviele Unterverzeichnisse es gibt und zeigt die Anzahl im fehlerfreien Fall am Ende an. Es ignoriert dabei alle Verzeichnisse, deren Namen mit einem Punkt beginnen – z.B. `/home/user/.ssh`. In dieser Aufgabe sollen Sie Teile des Programms implementieren.

The program `tree` prints the directory hierarchy, starting from the specified path. The program counts all encountered subdirectories and prints the number at the end if no error occurred. It ignores all directories, starting with a dot – e.g., `/home/user/.ssh`. Your objective is to implement some of the program's main functions.

Gewünschte Ausgabe von/*Desired output of:* `tree /home/user`

/home/user
-documents
--exams
-music
Directories: 3

d) Schreiben Sie die Funktion `isDir()`, die überprüft ob ein gegebener Verzeichniseintrag ein Unterverzeichnis ist, dessen Name nicht mit einem Punkt beginnt. Nehmen Sie an, dass das Programm nur auf ext2 Dateisystemen ausgeführt wird. **1 pt**

Implement the function `isDir()`, which checks if a given directory entry is a subdirectory, whose name does not start with a dot. Assume the program runs only on ext2 file systems.

e) Schreiben Sie die Funktion `enumDirs()`, die für einen gegebenen Pfad die Hierarchie der Unterverzeichnisse ausgibt und dabei den Verzeichniszähler aktualisiert. Nutzen Sie dabei die gegebenen Hilfsfunktionen. **6 pt**

Implement the function `enumDirs()`, which outputs the hierarchy of subdirectories for a given path and updates the directory counter. Use the supplied helper functions.

f) Schreiben Sie die Einsprungsfunktion `main()`. Das Programm soll mit dem aufzulistenden Verzeichnis als einzigen Parameter gestartet werden. Führen Sie eine Überprüfung der Parameter durch und rufen Sie anschließend die Funktion `enumDirs()` auf. Achten Sie auf die korrekte Ausgabe des Programms. **2 pt**

Implement the entry-point function `main()`. The program should be called with the directory to list as only parameter. Perform a parameter check and call the `enumDirs()` function. Pay attention to the correct output of the program.

```
int dirCount = 0;       // Directory counter
void printStats(void); // Prints the directory counter.

// Appends newDir to curDir and adds the path separator.
// Memory is handled internally - do not free the returned string.
// appendDir("home", "documents") => "home/documents"
char* appendDir(const char *curDir, const char *newDir);

// Prints a single directory with the supplied number of leading
// dashes to indicate the directory depth.
// printDir("home", 5) => "-----home\n"
void printDir(const char *dir, int depth);
```

```
// Returns 1, if dentry is a directory, 0 otherwise.
int isDir(const dirent *dentry) {
.....
.....
.....
.....
.....
.....
}

// Returns 0 on success, -1 otherwise.
int enumDirs(const char *dir, int depth) {
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
}
```

```
// Returns 0 on success, -1 otherwise.
int main(int argc, char *argv[]) {
```

}

Total: 12.0pt

Reserveplatz:/*Extra space:*

NAME

clock − determine processor time

SYNOPSIS

```
#include <time.h>

clock_t clock(void);
```

DESCRIPTION

The **clock**() function returns an approximation of processor time used by the program.

RETURN VALUE

The value returned is the CPU time used so far as a *clock_t*; to get the number of seconds used, divide by **CLOCKS_PER_SEC**. If the processor time used is not available or its value cannot be represented, the function returns the value *(clock_t) −1*.

CONFORMING TO

C89, C99, POSIX.1-2001. POSIX requires that **CLOCKS_PER_SEC** equals 1000000 independent of the actual resolution.

NOTES

Note that the time can wrap around. On a 32-bit system where **CLOCKS_PER_SEC** equals 1000000 this function will return the same value approximately every 72 minutes.

SEE ALSO

clock_gettime(2), **getrusage**(2), **times**(2)

COLOPHON

This page is part of release 3.65 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man−pages/.

NAME

errno − number of last error

SYNOPSIS

```
#include <errno.h>
```

DESCRIPTION

The <*errno.h*> header file defines the integer variable *errno*, which is set by system calls and some library functions in the event of an error to indicate what went wrong. Its value is significant only when the return value of the call indicated an error (i.e., −1 from most system calls; −1 or NULL from most library functions); a function that succeeds *is* allowed to change *errno*.

Valid error numbers are all nonzero; *errno* is never set to zero by any system call or library function.

For some system calls and library functions (e.g., **getpriority**(2)), −1 is a valid return on success. In such cases, a successful return can be distinguished from an error return by setting *errno* to zero before the call, and then, if the call returns a status that indicates that an error may have occurred, checking to see if *errno* has a nonzero value.

errno is defined by the ISO C standard to be a modifiable lvalue of type *int*, and must not be explicitly declared; *errno* may be a macro. *errno* is thread-local; setting it in one thread does not affect its value in any other thread.

All the error names specified by POSIX.1 must have distinct values, with the exception of **EAGAIN** and **EWOULDBLOCK**, which may be the same.

Below is a list of the symbolic error names that are defined on Linux. Some of these are marked *POSIX.1*, indicating that the name is defined by POSIX.1-2001, or *C99*, indicating that the name is defined by C99.

E2BIG Argument list too long (POSIX.1)

EACCES Permission denied (POSIX.1)

EADDRINUSE Address already in use (POSIX.1)

EADDRNOTAVAIL Address not available (POSIX.1)

EAFNOSUPPORT Address family not supported (POSIX.1)

EAGAIN Resource temporarily unavailable (may be the same value as **EWOULDBLOCK**) (POSIX.1)

EALREADY Connection already in progress (POSIX.1)

SEE ALSO

err(3), **error**(3), **perror**(3), **strerror**(3)

COLOPHON

This page is part of release 3.54 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man−pages/.

NAME

open − open and possibly create a file or device

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char ****pathname*, int** *flags***);**

DESCRIPTION

Given a *pathname* for a file, **open**() returns a file descriptor, a small, nonnegative integer for use in subsequent system calls (**read**(2), **write**(2), **lseek**(2), **fcntl**(2), etc.). The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

The argument *flags* must include one of the following *access modes*: **O_RDONLY**, **O_WRONLY**, or **O_RDWR**. These request opening the file read-only, write-only, or read/write, respectively.

In addition, zero or more file creation flags and file status flags can be bitwise-or'd in *flags*. The *file creation flags* are **O_CLOEXEC**, **O_CREAT**, **O_DIRECTORY**, **O_EXCL**, **O_NOCTTY**, **O_NOFOLLOW**, **O_TRUNC**, and **O_TTY_INIT**. The *file status flags* are all of the remaining flags listed below.

O_DIRECTORY
If *pathname* is not a directory, cause the open to fail. This flag is Linux-specific, and was added in kernel version 2.1.126, to avoid denial-of-service problems if **opendir**(3) is called on a FIFO or tape device.

ERRORS

EACCES
The requested access to the file is not allowed, or search permission is denied for one of the directories in the path prefix of *pathname*, or the file did not exist yet and write access to the parent directory is not allowed.

EFAULT
pathname points outside your accessible address space.

EISDIR
pathname refers to a directory and the access requested involved writing (that is, **O_WRONLY** or **O_RDWR** is set).

ENOTDIR
A component used as a directory in *pathname* is not, in fact, a directory, or **O_DIRECTORY** was specified and *pathname* was not a directory.

CONFORMING TO

SVr4, 4.3BSD, POSIX.1-2001. The **O_DIRECTORY**, **O_NOATIME**, **O_NOFOLLOW**, and **O_PATH** flags are Linux-specific, and one may need to define **_GNU_SOURCE** (before including *any* header files) to obtain their definitions.

SEE ALSO

chmod(2), **chown**(2), **close**(2), **dup**(2), **fcntl**(2), **link**(2), **lseek**(2), **mknod**(2), **mmap**(2), **mount**(2), **openat**(2), **read**(2), **socket**(2), **stat**(2), **umask**(2), **unlink**(2), **write**(2), **fopen**(3), **fifo**(7), **path_resolution**(7), **symlink**(7)

COLOPHON

This page is part of release 3.54 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man−pages/.

NAME

opendir, closedir − open/close a directory
readdir − read a directory

SYNOPSIS

#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char ****name*);**
struct dirent *readdir(DIR ****dirp*);**
int closedir(DIR ****dirp*);**

DESCRIPTION

The **opendir**() function opens a directory stream corresponding to the directory *name*, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

The **closedir**() function closes the directory stream associated with *dirp*. A successful call to **closedir**() also closes the underlying file descriptor associated with *dirp*. The directory stream descriptor *dirp* is not available after this call.

The **readdir**() function returns a pointer to a *dirent* structure representing the next directory entry in the directory stream pointed to by *dirp*. It returns NULL on reaching the end of the directory stream or if an error occurred.

On Linux, the *dirent* structure is defined as follows:

```
struct dirent {
    ino_t          d_ino;       /* inode number */
    unsigned char  d_type;      /* type of file; DT_DIR and DT_REG supported by ext2 fs */
    char           d_name[256]; /* filename */
};
```

The data returned by **readdir**() may be overwritten by subsequent calls to **readdir**() for the same directory stream.

RETURN VALUE

The **opendir**() functions return a pointer to the directory stream. On error, NULL is returned, and *errno* is set appropriately.

On success, **readdir**() returns a pointer to a *dirent* structure. (This structure may be statically allocated; do not attempt to **free**(3) it.) If the end of the directory stream is reached, NULL is returned and *errno* is not changed. If an error occurs, NULL is returned and *errno* is set appropriately.

ERRORS

EACCES
Permission denied.

ENOENT
Directory does not exist, or *name* is an empty string.

ENOTDIR
name is not a directory.

CONFORMING TO

opendir() is present on SVr4, 4.3BSD, and specified in POSIX.1-2001.

NAME

pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock − lock and unlock a mutex

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

DESCRIPTION

The mutex object referenced by *mutex* shall be locked by calling *pthread_mutex_lock*(). If the mutex is already locked, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner.

The *pthread_mutex_trylock*() function shall be equivalent to *pthread_mutex_lock*(), except that if the mutex object referenced by *mutex* is currently locked (by any thread, including the current thread), the call shall return immediately.

The *pthread_mutex_unlock*() function shall release the mutex object referenced by *mutex*. If there are threads blocked on the mutex object referenced by *mutex* when *pthread_mutex_unlock*() is called, resulting in the mutex becoming available, the scheduling policy shall determine which thread shall acquire the mutex.

If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the thread shall resume waiting for the mutex as if it was not interrupted.

RETURN VALUE

If successful, the *pthread_mutex_lock*() and *pthread_mutex_unlock*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

The *pthread_mutex_trylock*() function shall return zero if a lock on the mutex object referenced by *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

COPYRIGHT

NAME

sched_yield − yield the processor

SYNOPSIS

```
#include <sched.h>

int sched_yield(void);
```

DESCRIPTION

The *sched_yield*() function shall force the running thread to relinquish the processor until it again becomes the head of its thread list. It takes no arguments.

RETURN VALUE

The *sched_yield*() function shall return 0 if it completes successfully; otherwise, it shall return a value of -1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, <*sched.h*>

COPYRIGHT