Übungsblatt 14

Dr. Matthias Frank, Dr. Matthias Wübbeling

Ausgabe Mittwoch, 24. Januar 2024
Abgabe bis
Freitag, 2. Februar 2024, 23:59 Uhr
Dieses Übungsblatt wird nicht vorgeführt. Es dient lediglich der Klausurvorbereitung und muss auch nicht bearbeitet werden.

Die Aufgaben sind Klausuraufgaben aus vorangegangenen Klausuren, die Sie in einer Klausur lediglich mit einem Stift sowie der am Ende beigefügten Signaturübersicht bearbeiten können sollten. Zur Orientierung der Bearbeitungszeit können Sie sich die originalen Punktzahlen angucken. Für jeden Punkt sollten Sie ungefähr eine Minute brauchen; eine Aufgabe mit zehn Punkten sollte also ungefähr zehn Minuten benötigen.

Hinweis:	Die Seitenzahlen sowie die Nummern der Aufgaben sind aus den Klausuren übernommen und daher nicht korrekt.

Für jede der vier Aufgaben können Sie bis zu zwei Bonuspunkte erhalten, die wir auf Ihre Punkte für die Zulassung draufrechnen.

Aufgabe 2: (I/O-Multiplexing in C, 14 Punkte [6 x 1 + (4 x 1 + 2 x 2)])

In der Vorlesung haben Sie die Hilfsfunktion `select()` kennen gelernt. `select()` stellt universelle Funktionalität für I/O-Multiplexing mehrerer Eingabequellen zur Verfügung. `select()` läßt sich sowohl als blockierender als auch nicht-blockierender Aufruf realisieren.

Informationen für Select:

```
struct timeval {
  long tv_sec  /*seconds*/
  long tv_usec /*microseconds*/
};

int select(int n, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout)
```

a) (6 Punkte) Erläutern Sie kurz die Parameter und den Rückgabewert von `select()`:

```
int n
```

```
fd_set *readfds
```

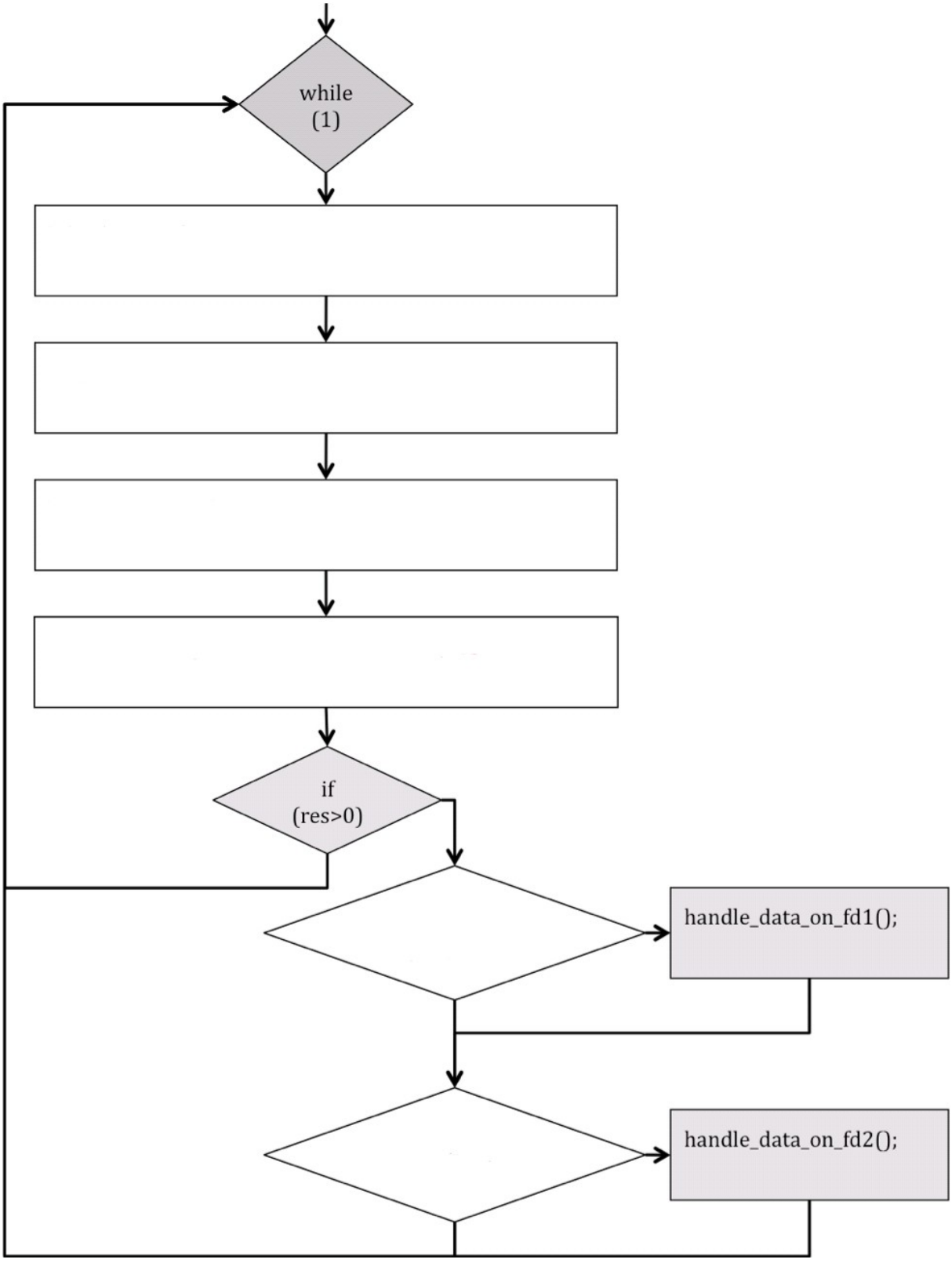
```
fd_set *writefds
```

```
fd_set *exceptfds
```

```
struct timeval *timeout
```

```
int select() – Rückgabewert:
```

b) (6 Punkte) Im Folgenden ist ein Ablaufschema für select mit zwei File-Descriptoren, fd1 und fd2, sowie die Set fd_set gegeben. Auf fd1 und fd2 soll lesend zugegriffen werden und der select -Aufruf soll blockierend sein. Füllen Sie die Kästen mit validem C-Code, so dass select zum I/O-Multiplexing zwischen den beiden File Descriptoren genutzt wird.



Aufgabe 3: (Assembler, 15 Punkte [3+6+2+4])

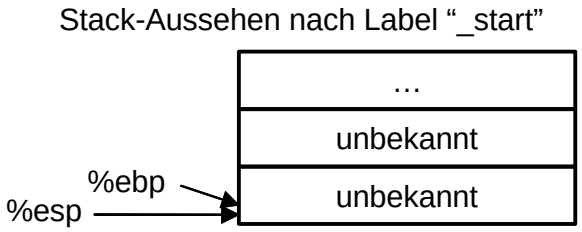
Gegeben ist das im Folgenden dargestellte Assembler-Programm für einen 32-Bit x86-Prozessor. Es besteht aus einer Hauptroutine und einer Unterroutine `getMax` mit der C-Signatur `int getMax(int *array, unsigned int size).`

Aus Gründen der Übersichtlichkeit ist das Programm auf diese und die folgenden Seiten verteilt.

Hauptroutine des Assemblerprogramms:

```
.data

     # Ein int-Array mit arraysize Elementen
array:
     .long 42, 4711, 23, 666, 34
     arraysize = 5

.text

.global _start

_start:

     # call getMax(&array, arraysize)
     pushl     $arraysize
     pushl     $array
     call      getMax
     # Parameter wieder vom Stack nehmen
     addl      $8, %esp

     # Prozess beenden
     movl      $1, %eax
     movl      $0, %ebx
     int       $0x80
```

Beim Eintritt in die Hauptroutine des Assemblerprogramms hat der Stack das folgende Aussehen (Zeitpunkt vor der Ausführung des ersten Befehls nach dem Label „_start“):



a) (3 Punkte) Ergänzen Sie nun den Stack (Belegung der Zellen mit Konstanten bzw. kurze Erklärung des Inhaltes), wie er nach dem Betreten der Unterroutine „`getMax`“ (siehe nächste Seite) beim Erreichen des Labels „`Zeitpunkt_1`“ aussieht. Kennzeichnen Sie auch die dann aktuelle Position des Basepointers `%ebp` und des Stackpointers `%esp`.

Platz für Ihre Lösung: **Reserve-Platz für Ihre Lösung:**

(bitte streichen Sie eine ungültige Lösung deutlich)

…
unbekannt
unbekannt

...
unbekannt
unbekannt

b) (6 Punkte) Die Unterroutine `int getMax(int *array, unsigned int size)` erwartet als Parameter einen Zeiger auf ein Array von Integer-Werten, mit `size` vielen Elementen. Sie soll das Maximum aller Array-Elemente zurückliefern und der in der Vorlesung besprochenen „cdecl Calling Convention“ gehorchen.
In das Programm haben sich **sechs** fehlerhafte Zeilen eingeschlichen. Finden Sie die sechs Zeilen, und korrigieren Sie die Fehler in den freien Feldern neben den Zeilen.

```
# int getMax(int *array, unsigned int size)
getMax:
    # Stack frame
    pushl    %ebp                  _____________________________
    movl     %esp, %ebp            _____________________________
Zeitpunkt_1:
    # 1 lokale Integer-Variable
    subl     $1, %esp              _____________________________
    # Rette Register (wg. cdecl)
    pushl    %ecx                  _____________________________
    movl     $0, -4(%ebp)          _____________________________
    # Hole Zeiger auf Array und Größe
    movl     12(%ebp), %esi        _____________________________
    movl     8(%ebp), %ecx         _____________________________

arrayloop:
    movl     (%esi), %eax          _____________________________
    cmpl     %eax, -4(%ebp)        _____________________________
    jnl      nichtkleiner          _____________________________
    movl     %eax, -4(%ebp)        _____________________________
nichtkleiner:
    addl     $4, (%esi)            _____________________________
    loop     arrayloop             _____________________________

    movl     -4(%ebp), %eax        _____________________________
    # Register wiederherstellen
    popl     %ecx                  _____________________________
    leave                          _____________________________
    ret                            _____________________________
```

c) (2 Punkte) Ersetzen Sie den Befehl „leave“ in den folgenden Zeilen durch eine äquivalente Folge von Assemblerbefehlen, wobei eine beliebige Auswahl aus den folgenden Befehlen erlaubt ist: {movl, pushl, popl, addl, subl, incl, decl}

Lösung:

_____

_____

_____

_____

d) (4 Punkte) In einem typischen Assemblerprogramm tauchen verschiedene Arten der Adressierung auf. Wie nennt man die folgenden Adressierungsarten?

`movl $1, %eax`	
`movl (1000), %eax`	
`movl (%ecx), %eax`	
`movl 8(%ebp), %eax`	

Aufgabe 3: (Busy Waiting in Threads, 13 Punkte [1 + 12])

Gegeben sei das folgende Programmfragment, in dem ein Thread in der Funktion „waitingThread“ so lange per Busy Waiting wartet, bis ein anderer Thread mit Hilfe der Funktion „setFlag“ einen Flagwert größer als Null setzt.

Hinweis: Am Ende der Klausur finden Sie eine Liste mit der Syntax relevanter Bibliotheksfunktionen.

```
int threadFlag = 0;
pthread_mutex_t threadFlagMutex;

void* waitingThread (void* thread_arg) {
     int localFlag = 0;
     while (!localFlag) {
          pthread_mutex_lock(&threadFlagMutex);
          localFlag = threadFlag;
          pthread_mutex_unlock(&threadFlagMutex);
     }
     printf("Condition fulfilled!\n");
     return 0;
}

void setFlag (int newFlagValue) {
     pthread_mutex_lock(&threadFlagMutex);
     threadFlag = newFlagValue;
     pthread_mutex_unlock(&threadFlagMutex);
}
```

a) (1 Punkt) Was ist der Hauptnachteil bei der Verwendung von Busy Waiting?

b) (12 Punkte) Ändern Sie die beiden Funktionen „waitingThread“ und „setFlag“ so ab, dass kein Busy Waiting mehr verwendet wird, sondern der in der Vorlesung diskutierte alternative Ansatz. Die Semantik des Programms soll ansonsten unverändert bleiben.

```
int threadFlag = 0;



void* waitingThread (void* thread_arg) {









      printf("Condition fulfilled!\n");
      return 0;
}

void setFlag (int newFlagValue) {









}
```

Aufgabe 4: (Thread-Synchronisation, insbes. Mutex, Semaphor, 20 Punkte + ggf. 2 Bonuspunkte [5 + 5 + 4 + 2 Bonus + 6])

a) (5 Punkte) In der Vorlesung wurden die folgenden fünf Konzepte für Thread-Synchronisation im Detail besprochen. Nennen Sie kurz und prägnant den Einsatzzweck bzw. charakteristische Eigenschaften des jeweiligen Konzeptes:

1. Mutex Variablen

_____

_____

2. Condition Variablen

_____

_____

3. Semaphore

_____

_____

4. Barriers

_____

_____

5. Read-write Locks

_____

_____

Betrachten Sie bei den weiteren Aufgabenteilen b) bis e) die folgenden gegebenen Funktionen, die allein mit Hilfe von Mutex-Variablen Semaphore realisieren - konkret die drei wesentlichen Funktionen für Semaphore, nämlich SEM_init(), SEM_wait() und SEM_post().
Auf eine Identifikation des Semaphors wird aus Gründen der besseren Lesbarkeit verzichtet, d.h. in einem Thread-fähigen Programm könnte nur eine Instanz unseres Semaphors benutzt werden.

```
int Sem_counter ;  // globale Counter-Variable

void SEM_init(int counter_init) {
     mutex_init(&CounterMutex, NULL)
     mutex_init(&SemMutex, NULL)

     mutex_lock(&CounterMutex);
     Sem_counter = counter_init;
     mutex_unlock(&CounterMutex);
}

void SEM_wait() {
     int block = 0;

     mutex_lock(&CounterMutex);
     Sem_counter--;
     printf("counter-- = %d\n", Sem_counter);
     If (Sem_counter <=0) block = 1;
     mutex_unlock(&CounterMutex);

     If (block) {
          printf("Vor lock = %d\n", Sem_counter);
          mutex_lock(&SemMutex);
          printf("Nach lock = %d\n", Sem_counter);
     }
}

void SEM_post() {
     mutex_lock(&CounterMutex);
     Sem_counter++;
     mutex_unlock(&CounterMutex);

     printf("post! = %d\n", Sem_counter);
     mutex_unlock(&SemMutex);
          // unlock eines nicht gelockten Mutex
          // bewirkt keinen Schaden
}
```

Die hier gegebene Implementation von SEM_wait() realisiert das Prinzip von wait() mit negativem Zähler:

1) Dekrementiere den Zähler um 1
2) Wenn Zähler < 0, dann blockiere
3) Nutze die Ressource (hier: verlasse SEM_wait())

b) (5 Punkte) Skizzieren Sie die Ausgabe (siehe printf()-Statements in SEM_wait() und SEM_post()) für folgenden zeitlichen Ablauf von Semaphor-Operationen. Nehmen Sie dabei an, dass die Funktionen SEM_wait() und SEM_post() nicht aufgrund von Thread-Scheduling unterbrochen werden (Ausnahme: Blockieren des Mutex <SemMutex> ist möglich).

SEM_init(3)

SEM_wait()

SEM_wait()

SEM_wait()

SEM_wait()

SEM_wait()

SEM_post()

Schreiben Sie Ihre Lösung neben die Aufrufe oben oder hier unterhalb:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

c) (4 Punkte) Die in der Aufgabe gegebene Implementation von SEM_wait() realisiert das Prinzip von wait() mit negativem Zähler:

1) Dekrementiere den Zähler um 1
2) Wenn Zähler < 0, dann blockiere
3) Nutze die Ressource (hier: verlasse SEM_wait())

Der Vergleich auf den Zählerstand in SEM_wait() lautet jedoch:

```
If (Sem_counter <=0) block = 1;
```

Erläutern Sie, warum ein Vergleich auf „< 0“ nicht das gewünschte Ergebnis bringen würde! (Hinweis: bedenken Sie die printf()-Ausgaben aus Aufgabe b) auf dem Weg zu Ihrer Antwort)

d) **Zusatzfrage!!!** Durch beantworten dieser Frage können Sie bei dieser Aufgabe **2 zusätzliche Punkte** bekommen:

In welcher Hinsicht könnte die oben genannte Implementierung eines Semaphors durch Mutex-Variablen im realen Einsatz problematisch sein? Denken Sie an die quasi-parallele Ausführung von Threads und die mögliche Unterbrechung durch Thread-Scheduling.

e) (6 Punkte) Die gegebene Implementierung von SEM_wait() soll nun so verändert werden, dass das folgende Prinzip mit nicht-negativem Zähler für Semaphore realisiert wird:

1) Wenn Zähler = 0, dann blockiere
2) Dekrementiere den Zähler um 1
3) Nutze die Ressource (hier: verlasse SEM_wait())

Schreiben Sie SEM_wait() nun entsprechend um. printf()-Ausgaben wie im gegebenen Beispiel brauchen dabei **nicht** enthalten sein. SEM_init() und SEM_post() brauchen nicht verändert werden (und sind zur Übersichtlichkeit nochmals dargestellt).

Denken Sie an korrekte Realisierung des Vergleiches des Zählerstandes (vgl. Aufgabenteil c).

```
int Sem_counter ;  // globale Counter-Variable

void SEM_init(int counter_init) {
     mutex_init(&CounterMutex, NULL)
     mutex_init(&SemMutex, NULL)

     mutex_lock(&CounterMutex);
     Sem_counter = counter_init;
     mutex_unlock(&CounterMutex);
}

void SEM_wait() {
```

Platz für Ihre Lösung:

```
} // Ende von SEM_wait()

void SEM_post() {
     mutex_lock(&CounterMutex);
     Sem_counter++;
     mutex_unlock(&CounterMutex);

     mutex_unlock(&SemMutex);
          // unlock eines nicht gelockten Mutex
          // bewirkt keinen Schaden
}
```

Signaturen relevanter Bibliotheksfunktionen

int **pthread_create**(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void*), void *arg);
int **pthread_attr_init**(pthread_attr_t *attr);
int **pthread_attr_destroy**(pthread_attr_t *attr);
void **pthread_exit**(void *value_ptr);
int **pthread_join**(pthread_t thread, void **value_ptr);
int **pthread_detach**(pthread_t thread);
int **pthread_cancel**(pthread_t thread);
int **pthread_setcancelstate**(int state, int *oldstate);
int **pthread_setcanceltype**(int type, int *oldtype);
void **pthread_testcancel**(void);
int **pthread_mutex_init**(pthread_mutex_t *mutex, pthread_mutexattr_t *attr);
int **pthread_mutex_destroy**(pthread_mutex_t *mutex);
int **pthread_mutexattr_init**(pthread_mutexattr_t *attr);
int **pthread_mutexattr_destroy**(pthread_mutexattr_t *attr);
int **pthread_mutexattr_settype**(pthread_mutexattr_t *attr, int type);
int **pthread_mutexattr_gettype**(const pthread_mutexattr_t *attr, int *type);
int **pthread_mutex_lock**(pthread_mutex_t *mutex);
int **pthread_mutex_trylock**(pthread_mutex_t *mutex);
int **pthread_mutex_unlock**(pthread_mutex_t *mutex);
int **pthread_cond_init**(pthread_cond_t *cond, const pthread_condattr_t *attr);
int **pthread_cond_destroy**(pthread_cond_t *cond);
int **pthread_condattr_init**(pthread_condattr_t *attr);
int **pthread_condattr_destroy**(pthread_condattr_t *attr);
int **pthread_cond_wait**(pthread_cond_t *cond, pthread_mutex_t *mutex);
int **pthread_cond_timedwait**(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime);
int **pthread_cond_signal**(pthread_cond_t *cond);
int **pthread_cond_broadcast**(pthread_cond_t *cond);
int **sem_init**(sem_t *sem, int pshared, unsigned int value);
int **sem_destroy**(sem_t *sem);
int **sem_wait**(sem_t *sem);
int **sem_trywait**(sem_t *sem);
int **sem_post**(sem_t *sem);
sem_t ***sem_open**(const char *name, int oflag);
sem_t ***sem_open**(const char *name, int oflag, mode_t mode, unsigned int value);
int **sem_close**(sem_t *sem);
int **sem_unlink**(const char *name);
int **pthread_barrier_init**(pthread_barrier_t *restrict barrier, const pthread_barrierattr_t *restrict attr, unsigned count);
int **pthread_barrier_destroy**(pthread_barrier_t *barrier);
int **pthread_barrier_wait**(pthread_barrier_t *);
int **pthread_rwlock_init**(pthread_rwlock_t *rwlock, const pthread_rwlockattr_t *attr);
int **pthread_rwlock_destroy**(pthread_rwlock_t *rwlock);
int **pthread_rwlock_rdlock**(pthread_rwlock_t *rwlock);
int **pthread_rwlock_tryrdlock**(pthread_rwlock_t *rwlock);
int **pthread_rwlock_trywrlock**(pthread_rwlock_t *rwlock);
int **pthread_rwlock_unlock**(pthread_rwlock_t *rwlock);
pid_t **getpid**(void);
pid_t **getppid**(void);

int **system**(const char *command);
pid_t **fork**(void);
int **execl**(const char *path, const char *arg0, ... /*, (char *)0 */);
int **execv**(const char *path, char *const argv[]);
int **execle**(const char *path, const char *arg0, ... /*, (char *)0, char *const envp[]*/);
int **execve**(const char *path, char *const argv[], char *const envp[]);
int **execlp**(const char *file, const char *arg0, ... /*, (char *)0 */);
int **execvp**(const char *file, char *const argv[]);
void **exit**(int status);
void **abort**(void);
pid_t **wait**(int *stat_loc);
pid_t **waitpid**(pid_t pid, int *stat_loc, int options);
int **pipe**(int fd[2]);
int **dup**(int fd);
int **dup2**(int oldfd, int newfd);
FILE ***popen**(const char *cmdstring, const char *type);
int **pclose**(FILE *fp);
int **mkfifo**(const char *pathname, mode_t mode);
mqd_t **mq_open**(const char *name, int oflag);
mqd_t **mq_open**(const char *name, int oflag, mode_t mode, struct mq_attr *attr);
mqd_t **mq_getattr**(mqd_t mqdes, struct mq_attr *attr);
mqd_t **mq_setattr**(mqd_t mqdes, struct mq_attr *newattr, struct mq_attr *oldattr);
int **mq_send**(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio);
ssize_t **mq_receive**(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio);
int **mq_close**(mqd_t mqdes);
int **mq_unlink**(const char *name);
int **mq_notify**(mqd_t mqdes, const struct sigevent *notification);
int **kill**(pid_t pid, int signo);
int **raise**(int signo);
int **sigaction**(int signo, const struct sigaction *act, struct sigaction *oact);
int **sigprocmask**(int how, const sigset_t *set, sigset_t *oset);
int **pthread_sigmask**(int how, const sigset_t *set, sigset_t *oset);
int **sigpending**(sigset_t *set);
int **sigemptyset**(sigset_t *set);
int **sigfillset**(sigset_t *set);
int **sigaddset**(sigset_t *set, int signo);
int **sigdelset**(sigset_t *set, int signo);
int **sigismember**(const sigset_t *set, int signo);
int **pause**(void);
int **sigsuspend**(const sigset_t *sigmask);
int **sigwait**(const sigset_t *set, int *sig);
ssize_t **read**(int filedes, void *buf, size_t nbytes);
ssize_t **write**(int fildes, const void *buf, size_t nbyte);
int **close**(int filedes);
int **fcntl**(int fildes, int cmd, ...);
off_t **lseek**(int filedes, off_t offset, int whence);
int **ioctl**(int fildes, int request, ... /* arg */);
int **socket**(int *domain*, int *type*, int *protocol*);
int **bind**(int *sockfd*, const struct sockaddr **my_addr*, socklen_t *addrlen*);

uint16_t **htons**(uint16_t hvalue);
uint32_t **htonl**(uint32_t hvalue);
uint16_t **ntohs**(uint16_t hvalue);
uint32_t **ntohl**(uint32_t hvalue);
int **sendto**(int sockfd, const void *buff, size_t nbytes, int flags, const struct sockaddr *to, socklen_t tolen);
int select(int maxfdp1, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
int **recvfrom**(int sockfd, void *buff, size_t nbytes, int flags, struct sockaddr *from, socklen_t *fromlen);
in_addr_t **inet_addr**(const char *dotted);
char ***inet_ntoa**(struct in_addr network);
int **listen**(int sockfd, int backlog);
int **connect**(int sockfd, const struct sockaddr *serv_addr,socklen_t addrlen);
int **accept**(int sockfd, struct sockaddr *addr,socklen_t *addrlen);
int **close**(int sockfd);
int **shutdown**(int sockfd, int howto);
FD_ZERO (fd_set *set);
FD_SET (int fd, fd_set *set);
FD_CLR (int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);
int **getsockname**(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);