Упражнение 7

Прихващане на сигнал

```
#include <signal.h>
typedef void (*sighandler_t)(int)
```
- указател към функция с параметър int и резултат void

```
sighandler_t signal(int signum, sighandler_t handler)
```
- задава реакция при сигнал

```
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)
```
- задава и/или получава реакция при сигнал

```
struct sigaction {
  void     (*sa_handler)(int); - реакция при сигнал
  void     (*sa_sigaction)(int, siginfo_t *, void *); - реакция при сигнал
  sigset_t   sa_mask; - маска на сигнали, които се блокират
  int        sa_flags; - флагове (SA_SIGINFO)
  void     (*sa_restorer)(void); - не се ползва
}
```

`SIG_IGN` - игнориране
`SIG_DFL` - реакция по подразбиране

Сигнали

`SIGHUP 1 Term` - прекъсване на връзката с управляващия терминал
`SIGINT 2 Term` - "прекъсване" от клавиатурата (Ctrl+C)
`SIGQUIT 3 Core` - "излизане" от клавиатурата (Ctrl+\)
`SIGILL 4 Core` - недопустима инструкция
`SIGABRT 6 Core` - сигнал от abort()
`SIGFPE 8 Core` - препълване при операция с плаваща точка
`SIGKILL 9 Term` - сигнал за убиване
`SIGSEGV 11 Core` - недопустима операция с паметта
`SIGPIPE 13 Term` - писане в канал без читатели
`SIGALRM 14 Term` - сигнал от alarm()
`SIGTERM 15 Term` - сигнал за прекратяване
`SIGUSR1 10 Term` - потребителски сигнал 1
`SIGUSR2 12 Term` - потребителски сигнал 2
`SIGCHLD 17 Ign` - завършване на син
`SIGCONT 18 Cont` - продължаване ако е спрял
`SIGSTOP 19 Stop` - спиране на процес

Изпращане на сигнал

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig)
```
- изпраща сигнал

```
#include <unistd.h>
int pause(void)
```
- чака сигнал

```
#include <unistd.h>
unsigned int alarm(unsigned int seconds)
```
- планира изпращането на SIGALRM

Упражнение 8

IPC обект

ipc := { msg | shm | sem } - опашка | памет | семафор

Създаване (отваряне) на IPC обект

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/ipc.h>
int ipcget(key_t key, int ipcflg)
```
- създава (отваря) IPC обект

IPC_PRIVATE - ключ за опашка между родствени процеси
*ipc*flg:
IPC_CREAT - създава IPC обект
IPC_EXCL - при създаване, грешка EEXIST, ако съществува

```
int msgget(key_t key, int msgflg)
int shmget(key_t key, size_t size, int shmflg)
int semget(key_t key, int nsems, int semflg)
```

Структура за права и собственост

```
#include <sys/ipc.h>
struct ipc_perm {
 uid_t uid;
 gid_t gid;
 uid_t cuid;
 gid_t cgid;
 mode_t mode;
 ...
};
```
- собственик
- група на собственика
- създател
- група на създателя
- права на достъп

Структура за IPC обект

```
#include <sys/ipc.h>
struct ipcid_ds {
 struct ipc_perm ipc_perm;
 ...
};
```

Управление на IPC обект

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/ipc.h>
int ipcctl(int ipcid, int cmd, struct ipcid_ds *buf)
```
- управлява IPC обект

cmd:
IPC_STAT - получава информация за IPC обект

`IPC_SET` - променя собственика и правата на IPC обект
`IPC_RMID` - унищожава IPC обект

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf)
int shmctl(int shmid, int cmd, struct shmid_ds *buf)
int semctl(int semid, int semnum, int cmd, union semun arg)
```

Упражнение 9

Създаване (отваряне) на опашка за съобщения

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t key, int msgflg)
```
- създава (отваря) опашка за съобщения

`IPC_PRIVATE` - ключ за опашка между родствени процеси
`msgflg`:
`IPC_CREAT` - създава опашка
`IPC_EXCL` - при създаване, грешка `EEXIST`, ако съществува

Структура за права и собственост

```
#include <sys/ipc.h>
struct ipc_perm {
 uid_t uid;
 gid_t gid;
 uid_t cuid;
 gid_t cgid;
 mode_t mode;
 ...
};
```
- собственик
- група на собственика
- създател
- група на създателя
- права на достъп

Структура за опашка

```
#include <sys/msg.h>
struct msqid_ds {
 struct ipc_perm msg_perm;
 time_t msg_stime;
 time_t msg_rtime;
 time_t msg_ctime;
 msgqnum_t msg_qnum;
 msglen_t msg_qbytes;
 pid_t msg_lspid;
 pid_t msg_lrpid;
};
```
- време на последно изпращане
- време на последно получаване
- време на последна промяна
- брой на съобщенията в опашката
- максимален брой байтове в опашката
- последния процес, изпратил съобщение
- последния процес, получил съобщение

Изпращане на съобщение

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg)
```
- изпраща съобщение

`msgflg`:
`IPC_NOWAIT` - ако няма място, завършва с грешка `EAGAIN`

Получаване на съобщение

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int
msgflg)
```
- получава съобщение

msgflg:
IPC_NOWAIT - ако няма съобщение, завършва с грешка ENOMSG
MSG_NOERROR - отрязва дългите съобщения, ако не - завършва с грешка E2BIG

Структура на съобщение

```
struct msgbuf {
 long mtype;
 char mtext[];
};
```
- тип (>0)
- съобщение

Управление на опашка

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf)
```
- управлява опашката

cmd:
IPC_STAT - получава информация за опашката
IPC_SET - променя собственика и правата на опашката
IPC_RMID - унищожава опашката и събужда блокираните процеси с грешка EIDRM

Упражнение 10

Алокиране (създаване) на сегмент обща памет

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg)
```
- алокира сегмент обща памет

IPC_PRIVATE - ключ за обща памет между родствени процеси
shmflg:
IPC_CREAT - създава сегмент обща памет
IPC_EXCL - при създаване, грешка EEXIST, ако съществува

Структура за права и собственост

```
#include <sys/ipc.h>
struct ipc_perm {
 uid_t uid;
 gid_t gid;
 uid_t cuid;
 gid_t cgid;
 mode_t mode;
 ...
};
```
- собственик
- група на собственика
- създател
- група на създателя
- права на достъп

Структура за обща памет

```
#include <sys/shm.h>
struct shmid_ds {
 struct ipc_perm shm_perm;
 size_t shm_segsz;
 time_t shm_atime;
 time_t shm_dtime;
 time_t shm_ctime;
 pid_t shm_cpid;
 pid_t shm_lpid;
 msgqnum_t shm_nattch;
};
```
- размер на сегмента (байта)
- време на последно присъединяване
- време на последно отделяне
- време на последна промяна
- процеса създал общата памет
- последния процес, изпълнил shmat()/shmdt()
- брой на присъединяванията

Присъединяване на сегмент обща памет

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg)
```
- присъединява сегмент към адресното пространство

shmflg:
SHM_RND - подравнява адреса надолу
SHM_RDONLY - присъединява сегмента само за четене

Отделяне на сегмент обща памет

```
#include <sys/types.h>
#include <sys/shm.h>
int shmdt(const void *shmaddr)
```
- отделя сегмент обща памет

Управление на общата памет

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buf)
```
- управлява общата памет

`cmd`:
`IPC_STAT` - получава информация за общата памет
`IPC_SET` - променя собственика и правата на общата памет
`IPC_RMID` - маркира общата памет за унищожаване

Упражнение 11

Създаване (отваряне) на множество семафори

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t key, int nsems, int semflg)
```
- създава (отваря) множество от семафори

`IPC_PRIVATE` - ключ за семафор между родствени процеси
`msgflg`:
`IPC_CREAT` - създава семафор
`IPC_EXCL` - при създаване, грешка `EEXIST`, ако съществува

Структура за права и собственост

```
#include <sys/ipc.h>
struct ipc_perm {
 uid_t uid;
 gid_t gid;
 uid_t cuid;
 gid_t cgid;
 mode_t mode;
 ...
};
```
- собственик
- група на собственика
- създател
- група на създателя
- права на достъп

Структура за множество семафори

```
#include <sys/sem.h>
struct semid_ds {
 struct ipc_perm sem_perm;
 time_t sem_otime;
 time_t sem_ctime;
 unsigned short sem_nsems;
};
```
- време на последна операция
- време на последна промяна
- брой семафори в множеството

Операции върху семафори

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned nsops)
```
- изпълнява операции върху семафори

Структура за операция върху семафор

```
struct sembuf {
 unsigned short sem_num;
 short sem_op;
 short sem_flg;
```
- индекс (номер) на семафора
- операция
- флагове

```
};
```

`sem_op`:
`>0` - прибавя стойността към семафора
`=0` - чака 0
`<0` - изважда стойността от семафора

`sem_flg`:
`IPC_NOWAIT` - завършва веднага и не се блокира
`SEM_UNDO` - отмяна на операцията, при край на процеса

Управление на множество семафори

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```
`int` **`semctl`**`(int semid, int semnum, int cmd, union semun arg)` - управлява семафор

`cmd`:
`IPC_STAT` - получава информация за множеството семафори в arg.buf
`IPC_SET` - променя собственика и правата на множество семафори
`IPC_RMID` - унищожава множество семафори
`GETALL` - връща стойностите на semval за цялото множество
`GETVAL` - връща стойността на semval за semnum
`SETALL` - задава стойностите на semval за цялото множество
`SETVAL` - задава стойността на semval за semnum

Обединение за управление на семафори

```
#include <sys/sem.h>
union semun {
 int val; - стойност на SETVAL
 struct semid_ds *buf; - буфер за IPC_STAT и IPC_SET
 unsigned short *array; - масив за GETALL и SETALL
};
```