Przekazywanie argumentów

```
int main(int argc, char* argv[])
```

- Przykład

```
#include <stdio.h>
int main(int argc, char* argv[])
{
int i;
printf ("%d\n", argc);
for(i=0; i<argc; i++)
printf("argument %d:  %s\n", i, argv[i]);
}
```

Pliki - biblioteki

```
#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

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

Otwieranie / tworzenie pliku

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

int open(const char *pathname, int flags, mode_t
mode)
```

Parametry:

- `pathname` — nazwa pliku (w szczególności nazwa ścieżkowa),
- `flags` — tryb otwarcia:
  - O_WRONLY
  - O_RDONLY
  - O_RDWR
  - O_APPEND
  - O_CREAT
  - O_TRUNC
- `mode` — prawa dostępu

Odczyt pliku

```
ssize_t read(int fd, void *buf, size_t count)
```

Parametry:

- `fd` — deskryptor pliku, z którego następuje odczyt danych,
- `buf` — adres początku obszaru pamięci, w którym zostaną umieszczone odczytane dane,
- `count` — liczba bajtów do odczytu z pliku (nie może być większa, niż rozmiar obszaru pamięci przeznaczony na odczytywane dane).

Zapis pliku

```
ssize_t write(int fd, const void *buf, size_t count)
```

Parametry:

- `fd` — deskryptor pliku, do którego następuje zapis danych,
- `buf` — adres początku obszaru pamięci, zawierającego blok danych do zapisania
- `count` — liczba bajtów do zapisania w pliku

Odczyt całego pliku

```
while((n=read(fd, buf, 20)) > 0)
{ write(1, buf, n); }
```

Wskaźnik bieżącej pozycji

```
off_t lseek(int fd, off_t offset, int whence)
```

Parametry:

- `fd` — deskryptor pliku
- `offset` — wielkość przesunięcia
- `whence` — odniesienie
  - SEEK_SET
  - SEEK_END
  - SEEK_CUR

Zamykanie/usuwanie pliku

```
int close(int fd)

int unlink(const char *pathname)
```

Tworzenie procesów

```
pid_t fork(void)
```

- Przykład

```
#include <stdio.h>
#include <unistd.h>
    int main(){
        printf ("Begin\n");
        fork();
        printf ("End\n");
        return 0;
    }
```

Proces macierzysty i potomny

```
#include <stdio.h>
#include <unistd.h>
main(){
    if (fork()==0)
        printf ("Child\n");
    else
        printf ("Parent\n");
    return 0;
}
```

Identyfikatory procesów

```
pid_t getpid(void)

pid_t getppid(void)
```

- Napisz program tworzący proces macierzysty i potomny. Dla każdego z proesów podaj wartość PID i PPID.

Zakończenie procesu

```
void exit(int status)
```

- Parametr
  - `status` — kod wyjścia przekazywany procesowi macierzystemu
- Przykład: exit(7);

```
int kill(pid_t pid, int signum)
```

- Parametry
  - `pid` — identyfikator procesu, do którego adresowany jest sygnał
  - `signum` — numer przesyłanego sygnału
- Przykład: kill(pid, 9);

Status zakończenia

```
#include <sys/wait.h>

pid_t wait(int *status)
```

- Parametr
  - `status` — adres słowa w pamięci, w którym umieszczony zostanie status zakończenia
- Funkcja zwraca identyfikator zakończonego procesu lub -1 w przypadku błędu

- Jesli wywołanie funkcji `wait` nastąpi przed zakonczeniem potomka, przodek zostaje zawieszony w oczekiwaniu na to zakonczenie.
- Po zakończeniu potomka następuje wyjście procesu macierzystego z funkcji `wait`.
- Pod adresem wskazanym w parametrze znajduje się status zakończenia.

Status zakończenia

- Status zakończenia:
  - numer sygnału (mniej znaczące 7 bitów)
  - kod wyjścia (bardziej znaczący bajt będący wartością fn. `exit` wywołanej przez potomka)

```
int status;
...
if (fork()!=0)
wait(NULL)

...
if (fork()!=0)
wait(&status)

printf( %x, status)
printf( %04x, status)
```

Proces zombi / sierota

- Sierota — proces potomny, którego przodek się już zakończył
- Zombi — proces potomny, który zakończył swoje działanie i czeka na przekazanie statusu zakończenia przodkowi
  - System nie utrzymuje procesów zombi, jeśli przodek ignoruje sygnał SIGCLD
- Zadanie: Stwórz proces sierotę i proces zombi

Wykonanie programu

```
int execl(const char *path,const char *arg,...)

int execlp(const char *file,const char *arg,...)

int execle(const char *path,const char *arg ,..., char *const
envp[])

int execv(const char *path, char *const argv[])

int execvp(const char *file, char *const argv[])

int execve(const char *file, char *const argv[], char *const
envp[])
```

- Parametry:
  - `path` --- nazwa ścieżkowa pliku z programem,
  - `file` --- nazwa pliku z programem,
  - `arg` --- argument linii poleceń
  - `argv` --- wektor (tablica) argumentów linii poleceń
  - `envp` --- wektor zmiennych środowiskowych.

Wykonanie programu

- `execl ("/bin/ls", "ls", "-a", NULL);`
- `execlp("ls", "ls", "-a", NULL);`
- ```
  char *const av[]={"ls", "-a", NULL};
  execv ("/bin/ls", av);
  ```
- ```
  char *const av[]={"ls", "-a", NULL};
  execvp ("ls", av);
  ```

Łącza

- ograniczona liczba bloków z danymi — łącza mają rozmiar: 4KB - 8KB w zależności od konkretnego systemu
- dostęp sekwencyjny (nie ma możliwości przemieszczania wskaźnika bieżącej pozycji, nie wywołuje się fn. lseek)
- dane odczytane z łącza są z niego usuwane
- proces jest blokowany w fn. read na pustym łączu, jeśli jest otwarty jakiś deskryptor tego łącza do zapisu
- proces jest blokowany w fn. write, jeśli w łączu nie ma wystarczającej ilości wolnego miejsca do zapisania całego bloku.
- przepływ strumienia — dane są odczytywane w kolejności, w której były zapisane

Rodzaje łączy

- łącze nienazwane (potok) — nie ma nazwy w żadnym katalogu i istnieje tak długo po utworzeniu, jak długo otwarty jest jakiś deskryptor tego łącza.
- łącze nazwane (kolejka FIFO) — ma dowiązanie w systemie plików, co oznacza, że jego nazwa jest widoczna w jakimś katalogu i może ona służyć do identyfikacji łącza

Tworzenie łącza nienazwanego

```
#include <unistd.h>

int pipe(int fd[2])
```

- Parametr
  - `fd` — tablica 2 deskryptorów;
  - `fd[0]` — deskryptor potoku do odczytu
  - `fd[1]` — deskryptor potoku do zapisu
- Funkcja zwraca 0 w przypadku poprawnego zakończenia lub -1

Powielanie deskryptorów (II)

- `int dup2 (int oldfd, int newfd)`

- Parametr
  - `oldfd` — deskryptor który ma zostać powielony
  - `newfd` — numer nowoprzydzielonego deskryptora
- Powielenie (duplikacja deskryptora) we wskazanym miejscu w tablicy dskryptorów
- Funkcja zwraca numer nowo przydzielonego deskryptora lub -1 w przypadku błędu

Tworzenie łącza nazwanego

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

int mkfifo(const char *pathname, mode_t mode)
```

- Funkcja zwraca 0 w przypadku poprawnego zakończenia lub -1
- Parametr
  - `pathname` — nazwa pliku (w szczególności nazwa ścieżkowa)
  - `mode` — prawa dostępu do nowo tworzonego pliku.

Otwieranie łącza

```
mkfifo("kolFIFO", 0600);

open("kolFIFO", O_RDONLY);
```

- Funkcja `open` musi być wywołana w trybie komplementarnym
- Polecenie systemowe `mkfifo`

Obsługa sygnałów

1. Sygnały są obsługiwane w sposób asynchroniczny
2. Reakcja procesu na otrzymany sygnał:
   - Wykonanie akcji domyślnej (najczęściej zakończenie procesu z ewentualnym zrzutem zawartości segmentów pamięci na dysk)
   - Zignorowanie sygnału
   - Przechwycenie sygnału tj. podjęcie akcji zdefiniowanej przez użytkownika
3. Lista sygnałów: `kill -l`, `man 7 signal`

Wysłanie sygnału

```
#include <signal.h>

int kill(pid_t pid, int signum)
```

- Funkcja zwraca 0 w przypadku poprawnego zakończenia lub -1
- Parametry:
  - `pid` — identyfikator procesu, do którego adresowany jest sygnał
    - `pid > 0` sygnał zostanie wysłany do procesu o identyfikatorze pid,
    - `pid = 0` sygnał zostanie wysłany do grupy procesów do których należy proces wysyłający,
    - `pid = -1` sygnał zostanie wysłany do wszystkich procesów oprócz wysyłającego i procesu INIT,
    - `pid < -1` oznacza procesy należące do grupy o identyfikatorze `-pid`.
  - `signum` — numer przesyłanego sygnału

Wysłanie sygnału

```
int raise(int signo)
```

- Wysłanie przez proces sygnału do samego siebie

Obsługa sygnału

```
void *signal(int signum, void *f())
```

- Parametry:
  - `signum` numer sygnału, którego obsługa ma zostać zmieniona
  - `f` może obejmować jedną z trzech wartości:
    - `SIG_DFL` — (wartość 0) standardowa reakcja na sygnał
    - `SIG_IGN` — (wartość 1) ignorowanie sygnału
    - Wskaźnik do funkcji - wskaźnik na funkcję, która będzie uruchomiona w reakcji na sygnał
- Funkcja zwraca:
  - wskaźnik na poprzednio ustawioną funkcję obsługi (lub `SIG_IGN, SIG_DFL`)
  - `SIG_ERR` w wypadku błędu

Obsługa sygnału

- Nie można przechwytywać, ani ignorować sygnałów `SIGKILL` i `SIGSTOP`.
- Gdy w procesie macierzystym ustawiony jest tryb ignorowania sygnału `SIGCLD` to po wywołaniu funkcji `exit` przez proces potomny, proces zombi nie jest zachowywany i miejsce w tablicy procesów jest natychmiast zwalniane

Przykład użycia funkcji signal

```
void (*f)();

f=signal(SIGINT,SIG_IGN); //ignorowanie sygnału
SIGINT

signal(SIGINT,f); //przywrócenie poprzedniej reakcji
na syg.

signal(SIGINT,SIG_DFL); //ustaw.  standardowej
reakcji na syg.

void moja_funkcja() {
printf(Został przechwycony sygnał\n");
exit(0);
}

main(){
signal(SIGINT,moja_funkcja); //przechwycenie sygnału
}
```

Przykład użycia funkcji signal

```
void obsluga(int signo) {
printf("Odebrano sygnał %d\n", signo);
}

int main() {
signal(SIGINT, obsluga);
while(1) ; //pętla nieskończona
}
```

Oczekiwanie na sygnał

```
void *pause()
```

- Działanie:
  - Zawiesza wywołujący proces aż do chwili otrzymania dowolnego sygnału.
  - Najczęściej sygnałem, którego oczekuje pause jest sygnał pobudki `SIGALRM`.
  - Jeśli sygnał jest ignorowany przez proces, to funkcja `pause` też go ignoruje.

Funkcja alarm

```
unsigned alarm ( unsigned int sek )
```

- Parametry:
  - `sek` — ilość sekund po których wysyłany jest sygnał SIGALRM
- Działanie:
  - Funkcja wysyła sygnał `SIGALRM` po upływie czasu podanym przez użytkownika.
- Wynik:
  - Jeśli w momencie wywołania oczekiwano na dostarczenie sygnału zamówionego wcześniejszym wywołaniem funkcji, zwracana jest liczba sekund pozostała do jego wygenerowania.
  - W przeciwnym wypadku zwracane jest 0.

Funkcja `alarm`

- Uwaga:
  - Jeśli nie otrzymano jeszcze sygnału zamówionego wcześniejszym wywołaniem funkcji `alarm` poprzednie zamówienie zostaje unieważnione
  - Procesy wygenerowane przez funkcję fork() mają wartości swoich alarmów ustawione na 0
  - Procesy utworzone przez funkcję exec będą dziedziczyły alarm razem z czasem pozostałym do zakończenia odliczania