SEMAFORY NAZWANE

```
sem_t *sem_open(const char *name, int oflag,
                mode_t mode, unsigned int value);
```

- name — nazwa semafora, np. "/sem0"
- oflag — flagi: O_CREAT | O_RDWR
- mode — prawa dostępu, np. 0600
- value — wartość początkowa, np. 1

Przykład użycia:

```
sem_t *sem = sem_open("/counter", O_CREAT | O_RDWR, 0600,
1);
int sem_wait(sem_t *sem);    // P() - blokuje jeśli 0
int sem_post(sem_t *sem);    // V() - zwiększa o 1
int sem_close(sem_t *sem);   // zamyka w procesie
int sem_unlink(const char *name); // usuwa z systemu
```

PAMIĘĆ WSPÓŁDZIELONA

```
int shm_open(const char *name, int oflag, mode_t mode);
```

- name — nazwa segmentu, np. "/pamiec"
- oflag — O_CREAT | O_RDWR lub O_RDONLY
- mode — prawa dostępu, np. 0644

```
int ftruncate(int fd, off_t length); // ustawia rozmiar
void *mmap(void *addr, size_t length, int prot,
           int flags, int fd, off_t offset);
```

- addr — NULL (system wybiera adres)
- prot — PROT_READ | PROT_WRITE
- flags — MAP_SHARED (współdzielona)

Pełny przykład:

```
int fd = shm_open("/mem", O_CREAT | O_RDWR, 0644);
ftruncate(fd, 1024);
char *ptr = mmap(NULL, 1024, PROT_READ | PROT_WRITE,
                 MAP_SHARED, fd, 0);
// użycie ptr jak zwykłej tablicy
munmap(ptr, 1024);
shm_unlink("/mem");
```

WĄTKI (PTHREAD)

```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine)(void *),
                   void *arg);
```

Przykład:

```
void *funkcja_watku(void *arg) {
    printf("Wątek działa\n");
    return NULL;
}

pthread_t tid;
pthread_create(&tid, NULL, funkcja_watku, NULL);
pthread_join(tid, NULL); // czeka na zakończenie
```

MUTEXY

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

```
pthread_mutex_lock(&mutex);
// sekcja krytyczna
pthread_mutex_unlock(&mutex);
```

Dynamiczna inicjalizacja:

```
pthread_mutex_init(&mutex, NULL);
pthread_mutex_destroy(&mutex);
```

ZMIENNE WARUNKOWE

```
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

Oczekiwanie:

```
pthread_mutex_lock(&mutex);
while (!warunek) {
    pthread_cond_wait(&cond, &mutex);
}
pthread_mutex_unlock(&mutex);
```

Sygnalizowanie:

```
pthread_cond_signal(&cond);    // jeden wątek
pthread_cond_broadcast(&cond); // wszystkie wątki
```

KOLEJKI KOMUNIKATÓW

```
mqd_t mq_open(const char *name, int oflag,
              mode_t mode, struct mq_attr *attr);
```

- name — nazwa kolejki, np. "/queue"
- oflag — O_CREAT | O_RDWR, O_RDONLY, O_WRONLY
- attr — NULL dla domyślnych parametrów

Atrybuty kolejki:

```
struct mq_attr {
    long mq_flags;    // flagi (0 lub O_NONBLOCK)
    long mq_maxmsg;   // max liczba wiadomości
    long mq_msgsize;  // max rozmiar wiadomości
    long mq_curmsgs;  // aktualna liczba wiadomości
};
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);
```

Przykład użycia:

```
mqd_t mq = mq_open("/queue", O_CREAT | O_RDWR, 0644, NULL);
mq_send(mq, "Hello", 5, 0);
char buf[100];
mq_receive(mq, buf, 100, NULL);
mq_close(mq);
mq_unlink("/queue");
```

SOCKETY UDP

```
int socket(int domain, int type, int protocol);
```

- domain — AF_INET (IPv4)
- type — SOCK_DGRAM (UDP), SOCK_STREAM (TCP)
- protocol — 0

Serwer UDP:

```
int sock = socket(AF_INET, SOCK_DGRAM, 0);
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(1234);
addr.sin_addr.s_addr = INADDR_ANY;
bind(sock, (struct sockaddr*)&addr, sizeof(addr));

char buf[100];
struct sockaddr_in client;
socklen_t len = sizeof(client);
recvfrom(sock, buf, 100, 0, (struct sockaddr*)&client,
&len);
sendto(sock, "OK", 2, 0, (struct sockaddr*)&client, len);
```

Klient UDP:

```
int sock = socket(AF_INET, SOCK_DGRAM, 0);
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_port = htons(1234);
inet_pton(AF_INET, "127.0.0.1", &server.sin_addr);
sendto(sock, "Hello", 5, 0, (struct sockaddr*)&server,
sizeof(server));
```

SOCKETY TCP

Serwer TCP:

```c
int sock = socket(AF_INET, SOCK_STREAM, 0);
// bind() jak w UDP
listen(sock, 5); // max 5 połączeń w kolejce

struct sockaddr_in client;
socklen_t len = sizeof(client);
int client_sock = accept(sock, (struct sockaddr*)&client,
&len);

char buf[100];
recv(client_sock, buf, 100, 0);
send(client_sock, "Response", 8, 0);
close(client_sock);
```

Klient TCP:

```c
int sock = socket(AF_INET, SOCK_STREAM, 0);
// konfiguracja address jak w UDP
connect(sock, (struct sockaddr*)&server, sizeof(server));
send(sock, "Hello", 5, 0);
char buf[100];
recv(sock, buf, 100, 0);
close(sock);
```

PRZYDATNE FUNKCJE

```c
#include <arpa/inet.h>
inet_pton(AF_INET, "192.168.1.1", &addr.sin_addr); //
string → binary
char str[INET_ADDRSTRLEN];
inet_ntop(AF_INET, &addr.sin_addr, str,
INET_ADDRSTRLEN); // binary → string

htons(port);  // host → network byte order (short)
ntohs(port);  // network → host byte order (short)
```

KOMPILACJA

```
gcc -o program program.c -lpthread -lrt
```

- -lpthread — dla wątków
- -lrt — dla semaforów, pamięci współdzielonej, kolejek