

Programowanie Współbieżne

Sygnały czasu rzeczywistego
semafony
pamięć dzielona

Posix IPC

Sygnały w systemie Unix przez wiele lat ewaluowały

- Model sygnałów w wersji 7 systemu Unix (1978) był zawodny. Sygnały się gubiły i nie było możliwości zablokowania sygnałów.
- System 4.3BSD (1986) wprowadził sygnały niezawodne.
- System V 3.0 (1986) również wprowadził sygnały niezawodne ale w inny sposób
- Posix.1 (1990) ustandaryzował model BSD sygnałów niezawodnych.
- W ramach Posix.1 (1996) dodano sygnały czasu rzeczywistego.

Posix IPC

Sygnały obecnie możemy podzielić na

- Wszystkie znane nam już: SIGALRM, SIGINT, SIGTERM itd.. (0..31)
- Sygnały czasu rzeczywistego o wartościach od SIGRTMIN do SIGRTMAX Posix wymaga by było ich conajmniej RTSIG_MAX a wartość tam ma być minimum 8. W linuxie są to wartości 34-64.

Posix IPC

Sygnały czasu rzeczywistego cechy:

- Sygnały są kolejgowane
- Wielokrotne wystąpienie tego samego sygnału nie powoduje „zlepiania” w jeden sygnał
- Gdy do kolejki trafiają wielokrotne, nieblokowane sygnały czasu rzeczywistego to sygnały z niższym numerem mają wyższy priorytet.
- Sygnały RT przekazują nie tylko nr sygnału ale także: strukturę `siginfo_t` oraz kontekst.

Posix IPC

Procedura obsługi sygnału zdefiniowana jest następująco:

```
void func (int signo, siginfo_t * info, void *context);
```

- *signo* - numer sygnału

- *info* - struktura:

```
typedef struct {  
    int si_signo; /*numer sygnału*/  
    int si_errno; /* jeśli !=0 numer błędu związany z tym  
    sygnałem*/  
    int si_code; /*SI_{USER, QUEUE, TIMER, KERNEL,  
    ASYNCIO, MESGQ, SIGIO, TKILL, ASYNCLN} */  
    union {...}_sigfields; /* zależne od sygnału więcej w  
    siginfo.h dla naszych celów przyjmijmy union sigval  
    si_value;*/  
} siginfo_t;
```

- *context* - kontekst zależny od implementacji.

Posix IPC

Sygnały RT są generowane przez następujące zdarzenia i identyfikowane w polu ***si_code*** struktury ***siginfo_t***:

- ***SI_ASYNCIO*** – sygnał został wygenerowany przez zakończenie asynchronicznego zlecenia wejścia wyjścia (funkcje z serii *aio_**) np drivery USB
- ***SI_MESGQ*** – sygnał wygenerowany przy umieszczeniu komunikatu w pustej kolejce komunikatów
- ***SI_QUEUE*** - sygnał wysłany przez funkcję ***sigqueue***
- ***SI_TIMER*** – sygnał wygenerowany przez zakończenie pracy czasomierza ustawionego za pomocą ***timer_settim***
- ***SI_USER*** – sygnał wysłany przez funkcję ***kill***
- ***SI_TKILL*** – sygnał wysłany przez funkcję ***tkill***
- ***SI_KERNEL*** – sygnał wysłany przez funkcję jądra
- ***SI_SIGIO*** – sygnał wysłany przez SIGIO

Posix IPC

Aby określić procedurę obsługi sygnału używamy funkcji:

```
void sigaction(int numer_sygnału, struct sigaction *  
akcja, struct sigaction * poprzednia_akcja);
```

- **sigaction** – następująca struktura (opisana dalej):

```
struct sigaction {  
    void (*sa_handler)(int);  
    sigset_t sa_mask;  
    unsigned long sa_flags;  
    void (*sa_sigaction)(int, siginfo_t, void*)  
    void (*sa_restorer)(void);  
};
```

Posix IPC

Struktura **sigaction**:

- **sa_handler** – jeżeli sa_flags nie jest ustawione na SA_SIGINFO to jest to adres klasycznej funkcji obsługi sygnału. **void handler(int numer_sygnału);**
Możliwe też SIG_DFL, SIG_IGN dzięki którym przywracamy domyślną obsługę lub ignorujemy.
- **sa_mask** – dodatkowe sygnały do zablokowania
- **sa_flags** – flagi sygnału z serii SA_*:
 - **SA_NOCLDSTOP** – nie wysyłaj SIGCLD nawet gdy potomek się zatkończy
 - **SA_NOCLDWAIT** – nie twórz „zombie” po śmierci potomków.
 - **SA_SIGINFO** – użyj trzyargumentowej funkcji obsługi sygnału sa_sigaction zamiast sa_handler
 - **SA_NOMASK** – w czasie wykonywania sygnału nie jest on automatycznie blokowany
 - **SA_ONESHOT** – Kiedy sygnał jest wysłany, obsługa sygnału jest zerowana do SIG_DFL
 - **SA_RESTART** – Kiedy sygnał jest wysłany do procesu, w czasie, gdy ten wykonuje wolną funkcję systemową, funkcja systemowa jest wznowiana po powrocie z funkcji obsługi sygnału, tak powinno być dla wszystkich oprócz SIGALRM
- **sa_sigaction** – jeżeli SA_SIGINFO to określa adres funkcji obsługi sygnału
- **sa_restorer** – przywróć uchwyt? (podobno nie jest w specyfikacji Posix.1)

Posix IPC

Obsługa zbioru sygnałów ***sigset_t***:

int sigemptyset(sigset_t *set); - zeruje zbiór sygnałów

int sigfillset(sigset_t *set); - dodaje do zbioru wszystkie dostępne sygnały

int sigaddset(sigset_t *set, int sig); - dodaje do zbioru sygnał

int sigdelset(sigset_t *set, int sig); - usuwa ze zbioru sygnał

int sigismember(sigset_t *set, int sig); - Jeśli sygnał w jest w zbiorze zwróci != 0

Posix IPC

Obsługa zbioru sygnałów ***sigset_t***:

int sigpending(sigset_t *set); - Uzyskuje informacje, które z sygnałów są aktualnie niezałatwione

int sigsuspend(sigset_t *mask); - Wstrzymanie działania programu dopóki proces nie otrzyma sygnału. Na ten czas ustawia maskę sygnałów.
dostępne sygnały

void pause(); - wstrzymuje działanie procesu do nadejścia jakiegokolwiek sygnału

Posix IPC

Maska sygnałów (zestaw **sigset_t** sygnałów które mają być blokowane)

```
int sigprocmask(int how, sigset_t *setnew,  
                sigset_t *setold);
```

how – Co mamy zrobić z zestawem

- **SIG_BLOCK** – dodać **setnew** do listy blokowanych sygnałów
- **SIG_UNBLOCK** – odjąć **setnew** od listy blokowanych sygnałów
- **SIG_SETMAS** – ustawić **setnew** jako nową listę blokowanych sygnałów.

setnew – nowy zestaw

setold – stary zestaw

Poniższym wywołaniem można odczytać maskę sygnałów:

```
sigprocmask(SIG_BLOCK, NULL, &maska);
```

//przykład posix_signal.c

Posix IPC

Wysłanie sygnału

```
int sigqueue(pid_t pid, int sig, const union sigval value);
```

pid – komu chcemy wysłać sygnał

sig – nr sygnału

value – dodatkowe informacje

```
union sigval {  
    int sival_int;  
    void *sival_ptr;  
};
```

Posix IPC

WAŻNE!

Sygnały z założenia są narzędziem działającym asynchronicznie. Nie można robić żadnych założeń co do stanu wykonywanego programu. Należy pamiętać by w procedurze obsługi sygnału nie zmieniać globalnych danych. Jeżeli już to czynimy to tylko wtedy gdy inna część naszego oprogramowania z nich nie korzysta. Należy np. na moment użycia zmiennych globalnych, czy innych współdzielonych z procedurą obsługi sygnału zasobów zablokować sygnał, który tą obsługę mógł by wywołać.

Posix IPC

Semafony.

- Mogą być używane do synchronizowania procesów i wątków
- Semafony Systemu V
- Semafony Posiksowe nazwane
- Semafony Posiksowe w pamięci wspólnej.

Posix IPC - Semafor

- Stworzenie i inicjacja Semafora (zwykle poza wątkiem który będzie z niego korzystał).
- Operacja opuszczenia (wait,P)
- Operacja podniesienia (signal,V)

Posix IPC - Semafor

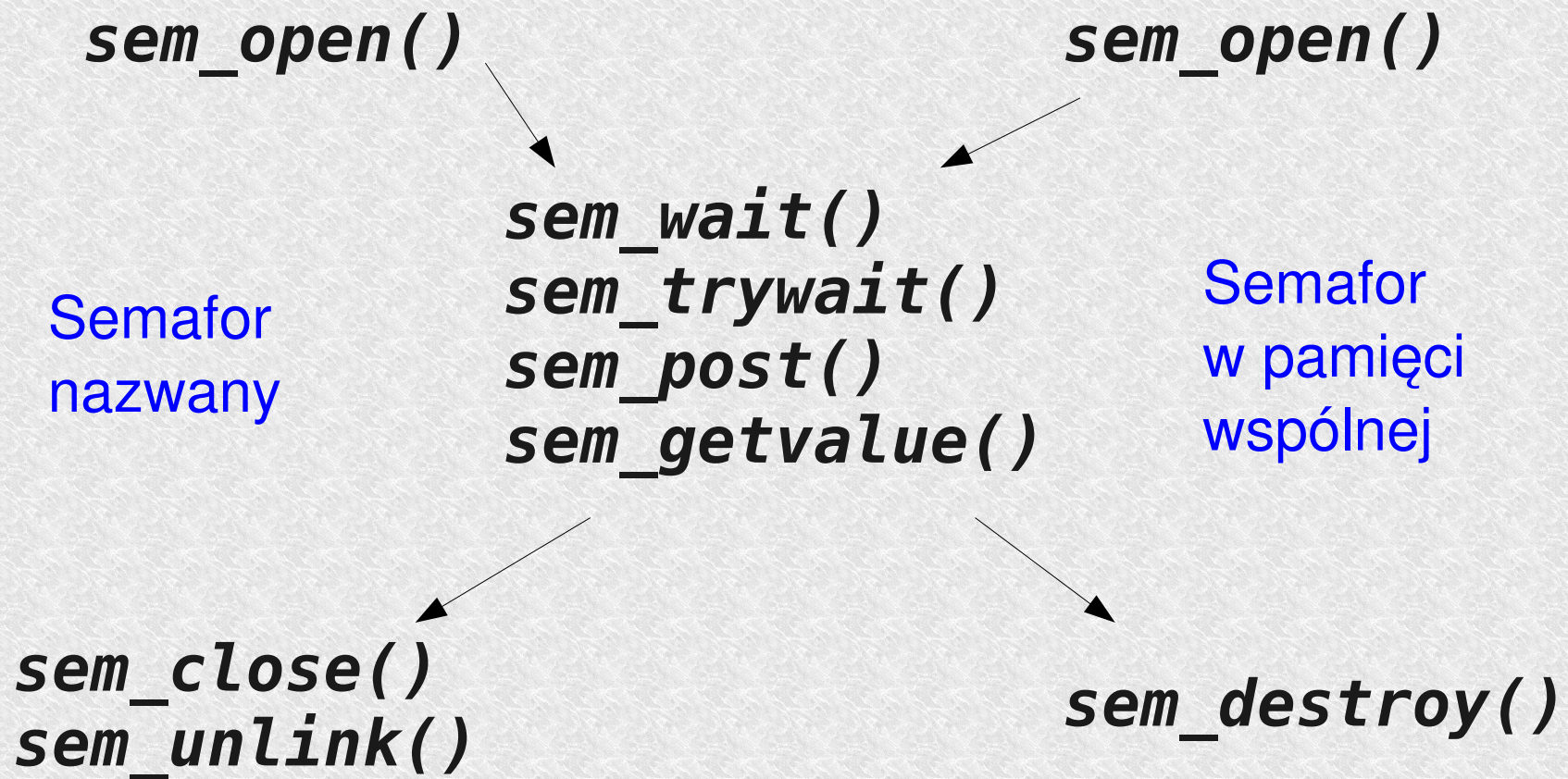
Wzajemne wykluczanie porównanie z mutexem.

```
pthread_mutex_lock(&mutex);  
//obszar krytyczny  
pthread_mutex_unlock(&mutex);
```

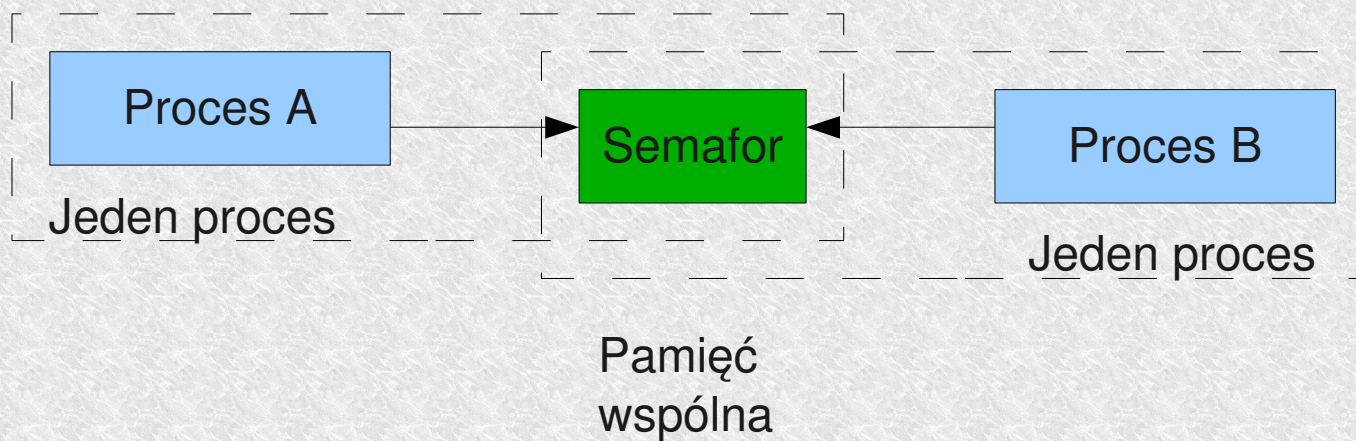
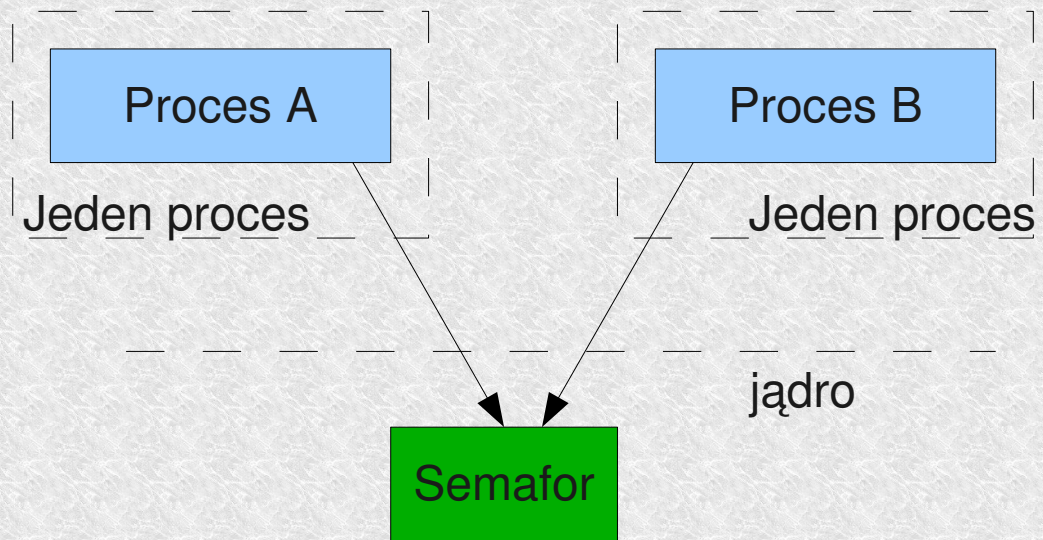
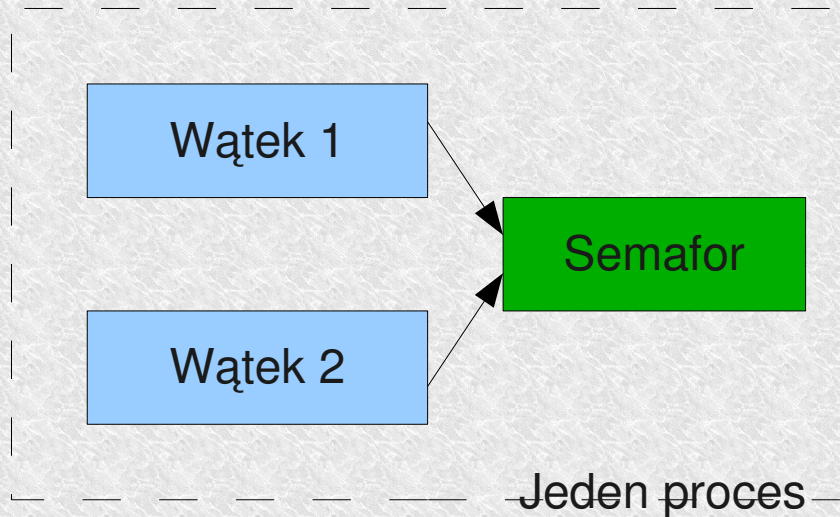
```
sem_wait(&sem);  
//obszar krytyczny  
sem_post(&sem);
```

Niemal identyczne zachowanie. Jednak odblokowanie mutexu może dokonać tylko wątek który go zablokował (tak jest w przypadku Windows), wywołanie odblokowania mutexu więcej niż 1 raz nie jest „pamiętane”, w przypadku mutexu w Windowsie wątek lokujący może wywoływać blokowanie wielokrotnie (blokowanie zagnieżdżone) semafor może odblokować drugi wątek. Ogólnie jednak semafony powinno się stosować do synchronizacji procesów a mutexów i zmienne warunkowe do synchronizacji wątków.

Posix IPC - Semafor



Posix IPC - Semafor



Posix IPC

Utworzenie / otworzenie semafora

```
#include <semaphore.h>
sem_t *sem_open(const char *name, int oflag, mode_t mode,
                unsigned int value);
```

- **name** – nazwa zgodna z normą posix (zaczyna się od „/”)
- **oflag** – czy tworzymy czy nie (O_CREAT | O_EXCL)
- **mode** – jeżeli tworzymy to podajemy bity uprawnień
- **value** – jeżeli tworzymy, wartość początkowa semafora
- Gdy ok zwróci wskaźnik do semafora gdy błąd zwróci wartość SEM_FAILED = -1.

Posix IPC

zamknięcie i usunięcie semafora

```
#include <semaphore.h>  
sem_t *sem_close(sem_t *sem);
```

- **sem** – semafor utworzony przez sem_open
- Gdy ok zwróci 0 gdy błąd -1.

```
#include <semaphore.h>  
sem_t *sem_unlink(const char *name);
```

- **name** – nazwa nadana podczas tworzenia semafora
- Gdy ok zwróci 0 gdy błąd -1.

Posix IPC

Oczekiwanie

```
#include <semaphore.h>  
sem_t *sem_wait(sem_t *sem);
```

- **sem** – semafor utworzony przez sem_open
- Gdy wartość semafora > 0 zmniejszy o 1 i będzie kontynuował
- Gdy wartość semafora 0 zostanie uśpiony do czasu gdy wartość semafora > 0.
- Gdy funkcja sem_wait skończy się przedwcześnie (np. Przez jakiś sygnał) zwrócone będzie EINTR
- Gdy błąd zwróci -1

```
#include <semaphore.h>  
sem_t *sem_trywait(sem_t *sem);
```

- **sem** – semafor utworzony przez sem_open
- Gdy wartość semafora > 0 zmniejszy o 1 i będzie kontynuował
- Gdy wartość semafora 0 zwróci EAGAIN
- Gdy błąd zwróci -1

Posix IPC

Sygnalizowanie i sprawdzanie wartości

```
#include <semaphore.h>
int sem_post(sem_t *sem);
```

- **sem** – semafor utworzony przez `sem_open`
- Gdy wartość semafora > 0 zwiększy ją o 1
- Gdy wartość semafora 0 zwiększy o jeden i obudzi jeden z wątków/procesów czekających na wartość > 0 .
- Gdy błąd zwróci -1 gdy ok zwróci 0.

```
#include <semaphore.h>
int sem_getvalue(sem_t *sem, int *valp);
```

- **sem** – semafor utworzony przez `sem_open`
- **valp** - bieżąca wartość semafora gdy < 0 to bezwzględna wartość oznacza ilość procesów czekających na danym semaforze.
- Gdy błąd zwróci -1 gdy ok 0.

Posix IPC

```
//sem_create.c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <semaphore.h>

int main (int argc, char **argv)
{
    int flags;
    sem_t *sem;
    unsigned int value;

    flags = O_RDWR | O_CREAT ;
    /* flags |= O_EXCL; */

    value = 1;
    if (0 > (sem = sem_open("/semafor", flags, 0666, value)))
    {
        perror("blad tworzenia semafora");
        exit(1);
    }
    printf("Ok semafor utworzony\n");
    sem_close(sem);
    printf("Ok semafor zamknięty\n");
    exit(0);
}
```


Posix IPC

```
//sem_getvalue.c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <semaphore.h>

int main (int argc, char **argv)
{
    sem_t *sem;
    int value;
    int flags = 0;

    flags = O_RDWR | O_CREAT ;
    /* flags |= O_EXCL; */

    value = 1;
    if (0 > (sem = sem_open("/semafor", flags, 0666, value)))
    {
        perror("blad otwierania semafora");
        exit(1);
    }
    printf("Ok semafor otwarty\n");
    sem_getvalue(sem, &value);
    printf("Value: %d\n", value);
    sem_close(sem);
    printf("Ok semafor zamknięty\n");
    exit(0);
}
```

Posix IPC

```
//sem_wait.c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <semaphore.h>

int main (int argc, char **argv)
{
    sem_t *sem;
    int value;
    int flags = 0;

    flags = O_RDWR | O_CREAT ;
    /* flags |= O_EXCL; */

    value = 1;
    if (0 > (sem = sem_open("/semafor", flags, 0666, value)))
    {
        perror("blad otwierania semafora");
        exit(1);
    }
    printf("Ok semafor otwarty\n");
    sem_wait(sem);
    printf("Ok semafor podniesiony\n");
    sem_getvalue(sem, &value);
    printf("Value: %d\n", value);
    sem_close(sem);
    printf("Ok semafor zamknięty\n");
    exit(0);
}
```

Posix IPC

```
//sem_signal.c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <semaphore.h>

int main (int argc, char **argv)
{
    sem_t *sem;
    int value;
    int flags = 0;

    flags = O_RDWR | O_CREAT ;
    /* flags |= O_EXCL; */

    value = 1;
    if (0 > (sem = sem_open("/semafor", flags, 0666, value)))
    {
        perror("blad otwierania semafora");
        exit(1);
    }
    printf("Ok semafor otwarty\n");
    sem_post(sem);
    printf("Ok semafor opuszczony\n");
    sem_getvalue(sem, &value);
    printf("Value: %d\n", value);
    sem_close(sem);
    printf("Ok semafor zamknięty\n");
    exit(0);
}
```

Posix IPC

```
//sem_unlink.c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <semaphore.h>

int main (int argc, char **argv)
{
    sem_unlink("/semafor");
    printf("Ok semafor usunięty\n");
    exit(0);
}
```

Posix IPC

Semaforey nienazwane

```
#include <semaphore.h>
```

```
int sem_init(sem_t *sem, int shared, unsigned int value);
```

- **sem** – semafor zaalokowany przez użytkownika
- **shared** - jeśli 0 to wspólny tylko dla wątków jeśli <> 0 to dla procesów i **sem** musi być w pamięci wspólnej.
- **value** - wartość początkowa semafora
- Gdy błąd -1 ale nie zwraca 0 gdy ok.

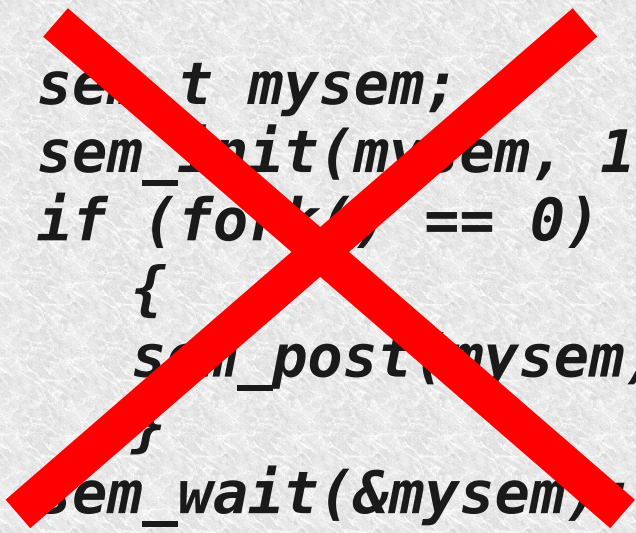
```
#include <semaphore.h>
```

```
int sem_destroy(sem_t *sem);
```

- **sem** – semafor zainicjowany sem_init
- Gdy ok zwróci 0 gdy błąd -1.

Posix IPC

- Należy zwrócić uwagę by `sem_init` wywoływać tylko raz
- Jeżeli `sem` przeznaczony jest do pamięci dzielonej trzeba uważać by nie operować na kopiach semaforów bo będą to różne wyniki.



```
sem_t mysem;  
sem_init(&mysem, 1, 0);  
if (fork() == 0)  
{  
    sem_post(&mysem);  
}  
sem_wait(&mysem);
```

Posix IPC – Pamięć wspólna

Funkcja **mmap** korzystamy:

- ze zwykłym plikiem, aby uzyskać wejście- wyjście oparte na odwzorowaniu w pamięci
- ze specjalnym plikiem, aby uzyskać anonimowe odwzorowanie pamięci
- z funkcją shm_open, aby uzyskać posixową pamięć wspólną, z której mają korzystać niezależne procesy

```
#include <sys/mman.h>
void *mmap(void *addr, size_t len, int prot, int flags, int
fd, off_t offset);
```

Jeśli ok przekazuje adres początkowy obszaru odwzorowania, jeśli błąd
MAP_FAILED

Posix IPC – Pamięć wspólna

- **addr** - może zawierać adres początkowy, na który ma być odwzorowany deskryptor **fd** w procesie. Zwykle = 0 wtedy system sam wybierze adres początkowy.
- **len** - liczba bajtów odwzorowanych w przestrzeni adresowej procesu.
- **offset** - określa pozycję w pliku, od której ma się zacząć odwzorowywanie, zwykle = 0
- **prot** - sposób ochrony:
 - PROT_READ – dane mogą być czytane
 - PROT_WRITE - zapisywane
 - PROT_EXEC - wykonywane
 - PROT_NONE – nie ma dostępu do danych
- **flags** - modyfikatory
 - MAP_SHARED – modyfikacje obszaru dokonane w procesie, który wywołał funkcję mmap, są widoczne przez wszystkie procesy, będące współwłaścicielami obiektu, i jednocześnie ulega zmianie obiekt przyporządkowany tej pamięci.
 - MAP_PRIVATE – wszelkie modyfikacje widoczne są tylko w procesie wywołującym funkcję mmap i nie powodują zmian w obiekcie odwzorowywanym.
 - MAP_FIXED – dokładna interpretacja argumentu addr
 - MAP_ANON – anonimowe odwzorowanie

Posix IPC – Pamięć wspólna

- Przy anonimowym odwzorowaniu `fd = -1` `MAP_SHARED` | `MAP_ANON`, offset jest pominięty, pamięć jest zerowana.
- Jeżeli chcemy mieć programy przenośne nie powinniśmy stosować `MAP_FIXED` a argument `addr` powinien mieć pusty wskaźnik.
- Aby współdzielić pamięć pomiędzy procesem potomnym a macierzystym można przed wywołaniem `fork` wywołać `mmap` z flagą `MAP_SHARED`. Posix.1 gwarantuje że odwzorowania pamięci z procesu macierzystego są dostępne w procesie potomnym, a zmiany w niej dokonywane są widoczne w obu procesach.

Posix IPC

Usuwanie odwzorowania:

```
#include <sys/mman.h>
int munmap(void *addr, size_t len);
```

- **addr** – adres, który otrzymaliśmy jako wynik funkcji mmap
 - **len** – rozmiar obszaru.
 - Gdy błąd -1 , 0 gdy ok.
-
- Po wywołaniu munmap wszelkie odwołania do danego obszaru pamięci będą się kończyły SIGSEGV
 - Gdy był ustawiony MAP_PRIVATE wszystkie zmiany są tracone
 - Gdy był ustawiony MAP_SHARED zmiany są przez jądro nanoszone na plik

Posix IPC

Synchronizowanie z plikiem, ręczne wywołanie

```
#include <sys/mman.h>  
int msync(void *addr, size_t len, int flags);
```

- **addr** – adres, który otrzymaliśmy jako wynik funkcji mmap
- **len** – rozmiar obszaru.
- **flags** – flagi
 - MS_ASYNC – realizuj zapis asynchronicznie (powrót od razu po zakolejkowaniu zadania przez jądro systemu)
 - MS_SYNC – realizuj zapis synchronicznie (powrót dopiero po wykonaniu zadania)
 - MS_INVALIDATE – unieważnij zapamiętane dane
- Gdy błąd -1 , 0 gdy ok.

Posix IPC

SHM

```
#include <sys/mman.h>
int shm_open(const char *name, int oflags, mode_t mode);
```

- **name** – posixsowa nazwa obiektu IPC
- **oflags** – flagi
 - O_RDONLY – tylko odczyt
 - O_RDWR – zapis odczyt
 - O_CREAT
 - O_EXCL
 - O_TRUNC – gdy jest z O_RDWR to istniejący obiekt zostanie skrócony do 0.
- - maska trybu dostępu, tylko ma znaczenie gdy użyto O_CREAT
- Gdy błąd -1 , gdy ok zwróci 0

Posix IPC

SHM - usunięcie

```
#include <sys/mman.h>  
int shm_unlink(const char *name);
```

- **name** – posixowa nazwa obiektu IPC
- Gdy błąd -1 , gdy ok zwróci 0

Posix IPC

Ustalenie rozmiaru pamięci wspólnej

```
#include <sys/mman.h>
int ftruncate(int *fd, off_t length);
```

- **fd** – deskryptor pliku
- **length** - wielkość w bajtach
 - Gdy zwykły plik to:
 - Jeśli rozmiar pliku był większy niż **length** to dodatkowe dane są kasowane
 - Jeśli był mniejszy nie wiadomo czy zawartość pliku się zmieni czy zostanie tylko zwiększona
 - Gdy obiekt pamięci wspólnej to funkcja ustali rozmiar obiektu = **length**
- Gdy błąd -1 , gdy ok zwróci 0

W celu sprawdzenia wielkości obiektu można użyć funkcji
`int fstat(int fd, struct stat * buf);` w strukturze stat jest pole `st_size`.

Posix IPC

```
//shm_create.c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <unistd.h>

int main (int argc, char **argv)
{
    int fd, flags;
    char *ptr;
    off_t length;

    flags = O_RDWR | O_CREAT ;
    /* flags |= O_EXCL; */

    length = 1024;
    if (-1 == (fd = shm_open("/pamiec_shm", flags, 0666)))
        perror("blad shm_open");
    if (-1 == (ftruncate(fd, length)))
        perror("blad ftruncate");
    if (MAP_FAILED == (ptr = mmap(NULL, length, PROT_READ | PROT_WRITE, MAP_SHARED,
                                   fd, 0)))
        perror("blad mmap");

    exit(0);
}
```

Posix IPC

```
//shm_write.c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int main (int argc, char **argv)
{
    int ii,fd;
    struct stat stat;
    unsigned char *ptr;

    if (-1 == (fd = shm_open("/pamiec_shm",O_RDWR, 0666)))
        perror("blad shm_open");
    if (-1 == (fstat(fd, &stat)))
        perror("blad fstat");
    if (MAP_FAILED == (ptr = mmap(NULL,stat.st_size, PROT_READ | PROT_WRITE,
                                MAP_SHARED, fd, 0)))
        perror("blad mmap");
    if (-1 == close(fd))
        perror("blad close");
    for (ii = 0;ii<stat.st_size;ii++)
        *ptr++=ii%256;
    exit(0);
}
```


Posix IPC

```
//shm_read.c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <unistd.h>

int main (int argc, char **argv)
{
    int ii,fd;
    struct stat stat;
    unsigned char c, *ptr;

    if (-1 == (fd =shm_open("/pamiec_shm",O_RDONLY,0666)))
        perror("blad shm_open");
    if (-1 == fstat(fd, &stat))
        perror("blad fstat");
    if (MAP_FAILED == (ptr = mmap(NULL,stat.st_size,PROT_READ,MAP_SHARED, fd, 0)))
        perror("blad mmap");
    if (-1 == close(fd))
        perror("blad mmap");
    //sprawdzenie czy faktycznie jest tak jak zapisaliśmy
    for (ii = 0; ii < stat.st_size; ii++)
    {
        c = *ptr++;
        if (!(ii%16)) printf("\n");
        if (!(ii%256)) printf("\n");
        printf("%.2X ",c);
        if (c != (ii%256))
            printf("Jednak nie są równe");
    }
    printf("\n");
    exit(0);
}
```


Posix IPC

```
//shm_unlink.c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>

int main (int argc, char **argv)
{
    if (!shm_unlink("/pamiec_shm"))
        printf("Ok nazwa obiektu usunięta\n");
    else
        perror("Błąd odlinkowywania pamięci");
    exit(0);
}
```