Systemy operacyjne

Laboratorium 6

Mateusz Małek
21 kwietnia 2017

Laboratorium 6

Kolejki komunikatów (System V i POSIX)

Komunikacja międzyprocesowa (IPC)

- Sygnały
- Potoki

- **Kolejki komunikatów**
  - Podobne do potoków? Nie do końca… Prawdę powiedziawszy, ani trochę.
  - Potok operuje na strumieniu bajtów, kolejka operuje na datagramach
  - Potok zachowuje się (po części) jak plik, kolejka ma dedykowane funkcje do obsługi
- Semafory
- Pamięć współdzielona

- Sockety

- *D-Bus, kdbus*
- *Binder (Android)*
- *Bus1*

Miejsce na żart o SR i CORBA...

IPC (System V)

- Żeby zobaczyć obiekty IPC SysV, można użyć komendy `ipcs` (man 1 ipcs)

```
[mkwm@temeraire:~] $ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 327681     mkwm       600        524288     2          dest
0x00000000 557059     mkwm       600        1048576    2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

- Żeby usunąć obiekty IPC SysV można użyć komendy `ipcrm` (man 1 ipcrm)
  - Trzeba podać klucz (`ipcrm -Q key`) lub ID obiektu (`ipcrm -q id`)
  - Przyda się w trakcie pracy nad zadaniami, jeśli coś pójdzie nie tak ;) (obiekty IPC nie znikają automatycznie po zakończeniu procesów)

Tworzenie obiektów IPC (System V)

- Potrzebny nam jest klucz
- Klucz to po prostu jakaś liczba
- Ludziom nie można ufać jak chodzi o wybór liczb (i nie tylko...)
  - Ludzie często wybierają liczby “okrągłe” lub w pewien sposób “szczególne”
  - Pewne wartości będą występowały częściej niż pozostałe
- Może lepiej taki klucz na podstawie czegoś wyznaczyć?
- key_t ftok(const char *pathname, int proj_id) - man 3 ftok
  - pathname musi dotyczyć istniejącego na dysku obiektu (np. katalog programu)
  - Z proj_id bierze się tylko 8 ostatnich bitów (czyli de facto podajemy tutaj pojedynczy char), musi być niezerowe - pozwala dla tego samego pathname uzyskać kilka obiektów IPC
  - Nie gwarantuje unikatowości, ale w praktyce - daje radę (Mieszanka numeru inode i numeru urządzenia blokowego)
- Alternatywa: jeśli nie musimy znać a priori, niech wygeneruje go system

Tworzenie kolejki (System V)

- int msgget(key_t key, int msgflg) - man 2 msgget
  - key to wartość zwrócona z ftok lub stała IPC_PRIVATE
  - W msgflag możemy podać uprawnienia do kolejki (jak w open) - jeśli jeszcze nie istniała i zostanie w tym momencie utworzona, to zostaną jej ustawione właśnie takie uprawnienia
  - Dostępne są też flagi IPC_CREAT i IPC_EXCL, analogiczne do flag O_CREAT i O_EXCL
  - Funkcja zwraca liczbowy identyfikator, jednoznacznie identyfikujący daną kolejkę w całym systemie
    - Jeśli użyliśmy IPC_PRIVATE, to wartość zwróconą przez msgget zwykle w jakiś sposób przekazujemy innemu procesowi (np. zapisując ją do innej, z góry ustalonej kolejki, która została utworzona w oparciu o ftok i ścieżkę na dysku)

Wysyłanie komunikatów (System V)

- msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg) - man 2 msgsnd
  - msqid to wartość zwrócona z msgget
  - msgp to wskaźnik do zdefiniowanej przez nas struktury postaci:

```
struct msgbuf {
    long mtype;
    // dowolne pola
}
```

    Na początku struktury obowiązkowo znajduje się pole mtype, ustawione na niezerową wartość - określające typ komunikatu. Dalej możemy umieścić dowolne pola - uwaga na tablice, w tym stringi - muszą mieć statyczny rozmiar; przesłanie do innego procesu wskaźnika nie ma sensu i nie zadziała!
  - msgsz to rozmiar zdefiniowanych przez nas pól struktury - nie obejmuje pola mtype! (czyli będzie to sizeof(struct msgbuf) - sizeof(long)); jeśli nasz komunikat nie zawiera żadnych danych (pól innych niż typ), to można i należy podać tutaj wartość 0
  - Flaga IPC_NOWAIT sprawia że funkcja natychmiast zwróci błąd, jeśli nie będzie miejsca w kolejce (domyślnie czeka na zwolnienie się miejsca)

Odbieranie komunikatów (System V)

- ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg) - man 2 msgrcv
  - msqid to wartość zwrócona z msgget
  - msgp to wskaźnik na zaalokowane miejsce, gdzie ma zostać zapisany odebrany komunikat
  - msgsz to rozmiar dostępnego miejsca na dane w obszarze wskazywanym przez msgp
  - msgtyp…
    - msgtyp == 0 - odbierz najdłużej oczekujący komunikat (pierwszy oczekujący w kolejce)
    - msgtyp > 0 - odbierz pierwszy oczekujący komunikat którego mtype == msgtyp… chyba że msgflg zawiera MSG_EXCEPT, wtedy warunek to mtype != msgtyp
    - msgtyp < 0 - odbierz pierwszy oczekujący komunikat o najniższym mtype mniejszym lub równym wartości bezwzględnej podanego msgtyp
  - msgflg pozwala nie czekać jeśli pożądanych komunikatów brak (IPC_NOWAIT), nie usuwać komunikatów z kolejki przy ich odbiorze (MSG_COPY), odwrócić zachowanie dla msgtyp > 0 (MSG_EXCEPT) lub przyciąć zbyt długie komunikaty do wielkości msgsz, zamiast zwracać błąd E2BIG (MSG_NOERROR)

Zarządzanie kolejką (System V)

- int msgctl(int msqid, int cmd, struct msqid_ds *buf) - man 2 msgctl
  - msqid to wartość zwrócona z msgget
  - cmd to pożądana akcja:
    - **IPC_RMID - natychmiastowe usunięcie kolejki i wybudzenie wszystkich czekających na nią procesów z errno EIDRM (argument buf jest ignorowany, można podać np. NULL)**
    - IPC_STAT - wypełnia strukturę msqid_ds wskazywaną przez buf aktualnymi informacjami na temat kolejki
    - IPC_SET - zmienia właściciela, uprawnienia i limit łącznej wielkości oczekujących komunikatów zgodnie z wartościami ustawionymi w strukturze msqid_ds wskazywanej przez buf

Struktury msqid_ds i ipc_perm (System V)

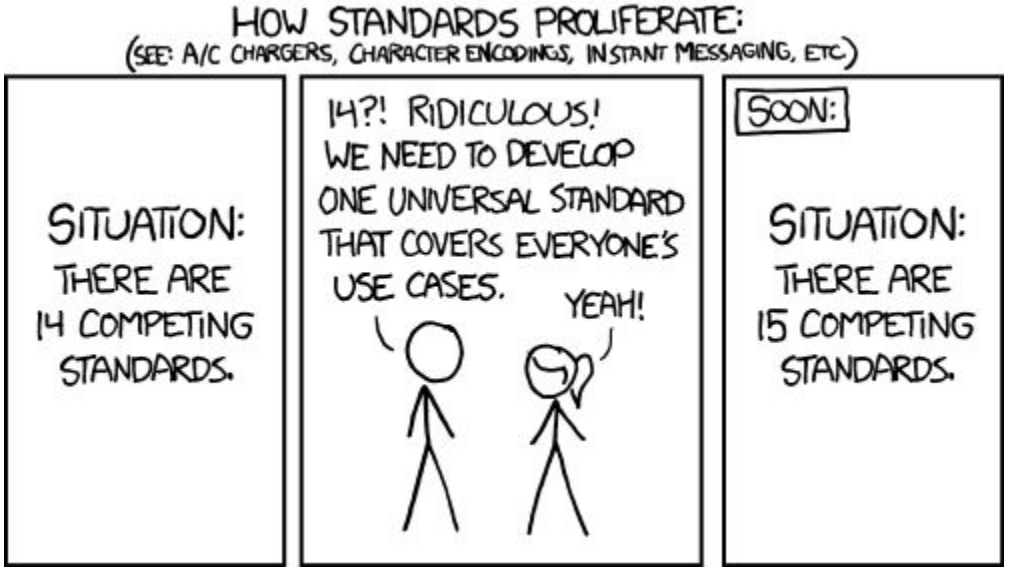
```
struct msqid_ds {
   struct ipc_perm msg_perm;     /* Ownership and permissions */
   time_t          msg_stime;    /* Time of last msgsnd(2) */
   time_t          msg_rtime;    /* Time of last msgrcv(2) */
   time_t          msg_ctime;    /* Time of last change */
   unsigned long   __msg_cbytes; /* Current number of bytes in
                                    queue (nonstandard) */
   msgqnum_t       msg_qnum;     /* Current number of messages
                                    in queue */
   msglen_t        msg_qbytes;   /* Maximum number of bytes
                                    allowed in queue */
   pid_t           msg_lspid;    /* PID of last msgsnd(2) */
   pid_t           msg_lrpid;    /* PID of last msgrcv(2) */
};
```

```
struct ipc_perm {
   key_t          __key;       /* Key supplied to msgget(2) */
   uid_t          uid;         /* Effective UID of owner */
   gid_t          gid;         /* Effective GID of owner */
   uid_t          cuid;        /* Effective UID of creator */
   gid_t          cgid;        /* Effective GID of creator */
   unsigned short mode;        /* Permissions */
   unsigned short __seq;       /* Sequence number */
};
```

Pogrubieniem zaznaczono wartości, które można modyfikować z wykorzystaniem komendy IPC_SET

IPC (POSIX)

Mała dygresja...



Fortunately, the charging one has been solved now that we've all standardized on mini-USB. Or is it micro-USB? Shit.

IPC (POSIX)

Do korzystania z POSIXowych mechanizmów IPC (kolejek, semaforów i pamięci wspólnej) konieczne jest zlinkowanie programu z biblioteką rt.

Mówiąc prościej - w pliku Makefile umieszczamy:

```
LDLIBS = -lrt
```

Tworzenie obiektów IPC (POSIX)

- Potrzebna nam jest jakaś nazwa
  - Wyjątek: semafory nienazwane (ale to nie na tym laboratorium)
- Nazwa ma postać stringa zaczynającego się od znaku /, nie dłuższego niż NAME_MAX (zakończonego oczywiście bajtem zerowym)
  - Jeśli coś wygląda jak ścieżka do pliku, ma ograniczenia jak ścieżka do pliku, to prawdopodobnie ma coś wspólnego z plikami

```
[mkwm:~] $ sudo mount -t mqueue mqueue /dev/mqueue
[mkwm:~] $ mount | grep mqueue
mqueue on /dev/mqueue type mqueue (rw,relatime,seclabel)
[mkwm:~] $ cd /dev/mqueue
[mkwm:/dev/mqueue] $ ls -l
total 0
-rw-rw-r--. 1 mkwm mkwm 80 01-02 03:04 sysopy
[mkwm:/dev/mqueue] $ cat sysopy
QSIZE:4          NOTIFY:0     SIGNO:0     NOTIFY_PID:0
```

  Na przykład w tym przypadku to podejrzenie ma pokrycie w rzeczywistości ;)

Tworzenie kolejki (POSIX)

- 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)
  - Jako name podajemy ścieżkę zgodną z omówionymi wcześniej wymaganiami
  - W oflag podajemy tryb pracy z kolejką (O_RDONLY/O_WRONLY/O_RDWR), w przypadku tworzenia kolejki można (i należy) podać flagę O_CREAT, dostępna jest też flaga O_EXCL lub O_NONBLOCK
  - W mode podajemy uprawnienia do kolejki (np. 0644)
  - W attr można podać wskaźnik do struktury opisującej parametry kolejki (niepodanie/przekazanie NULL sprawia, że kolejka utworzona zostanie z domyślnymi)
- Oba warianty opisane w man 3 mq_open

Struktura mq_attr (POSIX)

```
struct mq_attr {
    long mq_flags;       /* Flags: 0 or O_NONBLOCK */
    long mq_maxmsg;      /* Max. # of messages on queue */
    long mq_msgsize;     /* Max. message size (bytes) */
    long mq_curmsgs;     /* # of messages currently in queue */
};
```

- *kursywa* - z oczywistych względów nie można ustawić ;)
- **pogrubienie** - można ustawić tylko przy tworzeniu kolejki
- zwykły tekst - można ustawić przy tworzeniu kolejki lub zmienić później

Wysyłanie komunikatów (POSIX)

- int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio)
- int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio, const struct timespec *abs_timeout)
  - mqdes to deskryptor kolejki uzyskany z mq_open
  - msg_ptr to wskaźnik na komunikat do wysłania (niby char*, ale można zrzutować)
  - msg_len to rozmiar komunikatu wskazywanego przez msg_ptr
  - msg_prio to nieujemna liczba określają priorytet komunikatu - komunikaty z wyższą wartości msg_prio są zwracane przed komunikatami z niższą wartością tego parametru; komunikaty z identycznym msg_prio są zwracane w kolejności wysłania
  - abs_timeout pozwala określić maksymalny czas oczekiwania na zwolnienie się miejsca w zapełnionej kolejce (standardowo send się blokuje)
- Oba warianty opisane w man 3 mq_send

Odbieranie komunikatów (POSIX)

- ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio)
- ssize_t mq_timedreceive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio, const struct timespec *abs_timeout)
  - mqdes to deskryptor kolejki uzyskany z mq_open
  - msg_ptr to zaalokowane miejsce w pamięci, do którego zostanie zapisany komunikat
  - msg_len to rozmiar zaalokowanego miejsca w pamięci; musi być niemniejszy od atrybutu mq_msgsize danej kolejki
  - msg_prio to wskaźnik na int-a, w którym ma zostać zapisany priorytet zwróconego komunikatu (o ile jest różny od NULLa)
  - abs_timeout pozwala określić maksymalny czas oczekiwania na pojawienie się komunikatu w pustej kolejce (standardowo receive się blokuje)
- Oba warianty opisane w man 3 mq_receive

Powiadomienia o komunikatach (POSIX)

- int mq_notify(mqd_t mqdes, const struct sigevent *sevp) - man 3 mq_notify
  - mqdes to deskryptor kolejki uzyskany z mq_open
  - sevp to wskaźnik do struktury sigevent (man 7 sigevent); jeśli sevp == NULL, to "wyrejestrowujemy" się z powiadomienia
  - W danym momencie tylko jeden proces/wątek może być zarejestrowany na powiadomienia z danej kolejki. Rejestracja obowiązuje dla tylko jednego, najbliższego komunikatu.

```
struct sigevent {
    int          sigev_notify; /* Notification method */
    int          sigev_signo;  /* Notification signal */
    union sigval sigev_value;  /* Data passed with
                                  notification */
    void       (*sigev_notify_function) (union sigval);
                 /* Function used for thread
                    notification (SIGEV_THREAD) */
    void        *sigev_notify_attributes;
                 /* Attributes for notification thread
                    (SIGEV_THREAD) */
    pid_t        sigev_notify_thread_id;
                 /* ID of thread to signal (SIGEV_THREAD_ID) */
};
```

Powiadomienia o komunikatach cd. (POSIX)

- sigev_notify wskazuje w jaki sposób mamy zostać powiadomieni o nowych komunikatach:
  - SIGEV_SIGNAL - w momencie pojawienia się nowego komunikatu, wyślij do procesu sygnał:
    - sigev_signo wskazuje numer sygnału który ma zostać wysłany
    - sigev_value wskazuje jaką wartość powinien przenosić sygnał (na jaką wartość ma zostać ustawione si_value w strukturze opisującej sygnał)
  - SIGEV_THREAD - w momencie pojawienia się nowego komunikatu, uruchom nowy wątek:
    - sigev_notify_function wskazuje funkcję do uruchomienia
    - sigev_notify_attributes to (opcjonalne) atrybuty wątku (typu pthread_attr_t)
  - SIGEV_NONE - w momencie pojawienia się nowego komunikatu nic nie rób:
- Jeśli proces/wątek w danym momencie “wisi” na mq_receive, to nie zostanie do niego dostarczone powiadomienie (nie miałoby to większego sensu)

Zarządzanie atrybutami kolejki (POSIX)

- int mq_getattr(mqd_t mqdes, struct mq_attr *attr)
  - mqdes to deskryptor kolejki uzyskany z mq_open
  - attr to wskaźnik na strukturę w której mają zostać zapisane atrybuty kolejki
- int mq_setattr(mqd_t mqdes, const struct mq_attr *newattr, struct mq_attr *oldattr)
  - mqdes to deskryptor kolejki uzyskany z mq_open
  - newattr to nowe atrybuty kolejki do ustawienia (można zmienić tylko mq_flags)
  - oldattr to dotychczasowe atrybuty kolejki (przed zmianami)
- Obie funkcje opisane w man 3 mq_getattr

Zamknięcie kolejki (POSIX)

- int mq_close(mqd_t mqdes) - man 3 mq_close
  - mqdes to deskryptor kolejki uzyskany z mq_open
  - Kolejka zostaje zamknięta w danym procesie, ale nie jest usuwana z systemu
  - Ewentualna rejestracja na powiadomienie o komunikatach jest kasowana

Usuwanie kolejki (POSIX)

- int mq_unlink(const char *name) - man 3 mq_unlink
  - Jako name podajemy nazwę kolejki - nie deskryptor!
- Jeśli specjalny filesystem mqueue jest zamontowany (w /dev/mqueue), to można też po prostu usunąć plik reprezentujący kolejkę:

```
rm /dev/mqueue/sysopy
```

Dziękuję za uwagę