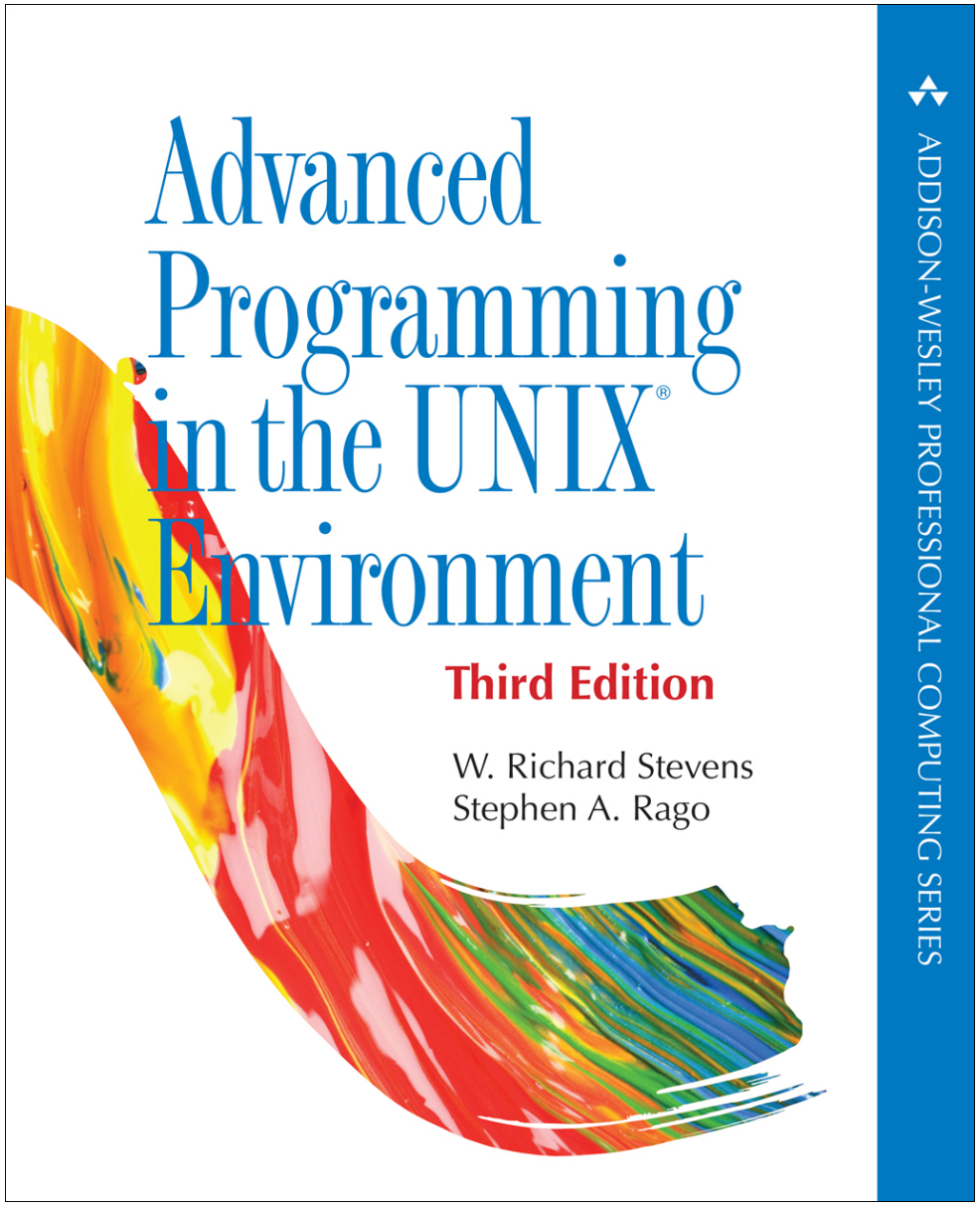
System V IPC

中正大學，作業系統實驗室
羅習五 陽春副教授



System V IPC

- 3 IPC
  - Message Queues
  - Semaphores
  - Shared Memory
- Originated in an internal AT&T version of UNIX called "Columbus UNIX"
- Later added to System V
- Criticized for inventing their own namespace instead of using the file system

XSI IPC

- Each IPC structure has a nonnegative integer identifier
- When creating an IPC structure, a key must be specified
  - Type: key_t
  - Defined in <sys/types.h> (long integer)
  - Converted into an identifier by kernel

Client-Server Rendezvous at same IPC structure (1)

- Server creates a new IPC structure using key = IPC_PRIVATE
  - ☘Guarantees new IPC structure
- Server stores returned identifier in some file for client to obtain
- Disadvantage: file I/O!

Client-Server Rendezvous at same IPC structure (2)

- Define a key in a common header
- Client and server agree to use that key
- Server creates a new IPC structure using that key
- Problem: key exists? (msgget, semget, shmget returns error)
- Solution: delete existing key, create a new one again

Client-Server Rendezvous at same IPC structure (3)

- Client and server agree on
  - a pathname
  - a project ID (char between 0 ~ 255)
- ftok() converts the 2 values into a key
- Client and server use that key (cf. 2)

ftok()

- #include <sys/types.h>
- #include <sys/ipc.h>
- key_t ftok(const char *pathname, int proj_id);

Permission Structure (Linux)

```
1.  struct ipc_perm {
2.    key_t __key; /* Key supplied to msgget(2) */
3.    uid_t uid; /* Effective UID of owner */
4.    gid_t gid; /* Effective GID of owner */
5.    uid_t cuid; /* Effective UID of creator */
6.    gid_t cgid; /* Effective GID of creator */
7.    unsigned short mode; /* Permissions */
8.    unsigned short __seq; /* Sequence number */
9.  };
```

Permission Structure (Linux)

- we can modify the uid, gid, and mode fields by calling msgctl, semctl, or shmctl.
- To change these values, the calling process must be either the creator of the IPC structure or the superuser.
- Chan(similar to calling chown or chmod for a file.
- mode

Permission	Bit
user-read	0400
user-write (alter)	0200
group-read	0040
group-write (alter)	0020
other-read	0004
other-write (alter)	0002

message queue

Message Queues

- Linked list of messages
- Stored in kernel
- Identified by message queue identifier
- msgget: create new or open existing queue
- msgsnd: send a msg to the queue
- msgrcv: receive msg from the queue
- Fetching order: based on type

msgget()

- #include <sys/types.h>
- #include <sys/ipc.h>
- #include <sys/msg.h>
- int msgget(key_t key, int msgflg);

msgget()

- The msgget() system call returns the System V message queue identifier associated with the value of the key argument.
- If **msgflg** specifies both IPC_CREAT and IPC_EXCL and a message queue already exists for key, then msgget() fails with errno set to EEXIST.

msgctl()

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

msgid_ds (Linux)

```
1.    struct msqid_ds {
2.      struct ipc_perm msg_perm; /* Ownership and permissions */
3.      time_t msg_stime; /* Time of last msgsnd(2) */
4.      time_t msg_rtime; /* Time of last msgrcv(2) */
5.      time_t msg_ctime; /* Time of last change */
6.      unsigned long __msg_cbytes; /* Current number of bytes in
7.        queue (nonstandard) */
8.      msgqnum_t msg_qnum; /* Current number of messages
9.        in queue */
10.     msglen_t msg_qbytes; /* Maximum number of bytes
11.       allowed in queue */
12.     pid_t msg_lspid; /* PID of last msgsnd(2) */
13.     pid_t msg_lrpid; /* PID of last msgrcv(2) */
14.   };
```

msgctl()

IPC_STAT	Fetch the msqid_ds structure for this queue, storing it in the structure pointed to by *buf*.
IPC_SET	Copy the following fields from the structure pointed to by *buf* to the msqid_ds structure associated with this queue: msg_perm.uid, msg_perm.gid, msg_perm.mode, and msg_qbytes. This command can be executed only by a process whose effective user ID equals msg_perm.cuid or msg_perm.uid or by a process with superuser privileges. Only the superuser can increase the value of msg_qbytes.
IPC_RMID	Remove the message queue from the system and any data still on the queue. This removal is immediate. Any other process still using the message queue will get an error of EIDRM on its next attempted operation on the queue. This command can be executed only by a process whose effective user ID equals msg_perm.cuid or msg_perm.uid or by a process with superuser privileges.

msgsnd() & msgrcv()

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

msgsnd() & msgrcv()

- (msgsnd) Specifying IPC_NOWAIT causes msgsnd to return immediately with an error of EAGAIN.
- (msgrcv) If the returned message is larger than *nbytes* and the MSG_NOERROR bit in *flag* is set, the message is truncated.

msgsnd() & msgrcv()

The *type* argument lets us specify which message we want.

- *type* == 0
  - The first message on the queue is returned.
- *type* > 0
  - The first message on the queue whose message type equals *type* is returned.
- *type* < 0
  - The first message on the queue whose message type is the lowest value less than or equal to the absolute value of *type* is returned.

msgsnd() & msgrcv()

- The msgp argument is a pointer to a caller-defined structure of the following general form:

```
1.  struct msgbuf {
2.    long mtype; /* message type, must be > 0 */
3.    char mtext[0~512]; /* message data, of length
4.      nbytes*/
5.  };
```

- The mtext field is an array (or other structure) whose size is specified by msgsz, a nonnegative integer value.
- Messages of zero length (i.e., no mtext field) are permitted.

semaphore

Semaphores

- A counter to provide access to shared data object for multiple processes
- To obtain a shared resource:
  - Test semaphore controlling resource
  - If value > 0, value--, grant use
  - If value == 0, sleep until value > 0
- Release resource, value++, sleeping processes waiting for the semaphore is awakened

XSI Semaphores

- A semaphore is defined as a set of one or more semaphore values. When we create a semaphore, we specify the number of values in the set.
- Creation (semget) is independent of initialization (semctl)
- All IPCs exist even if no process is using them.
  - Need to worry about process terminating without releasing semaphore.

semget

- #include <sys/types.h>
- #include <sys/ipc.h>
- #include <sys/sem.h>
- int semget(key_t key, int nsems, int semflg);

- The *semget*() returns the System V semaphore set identifier associated with the argument key. A new set of nsems semaphores is created if
  - key has the value IPC_PRIVATE or
  - if no existing semaphore set is associated with key and IPC_CREAT is specified in *semflg*.
- When a new set is created, the *semid_ds* structure are initialized.
- If we are referencing an existing set (a client), we can specify *nsems* as 0.
- flag: IPC_CREAT and IPC_EXCL

semid_ds (Linux)

```
1.  struct semid_ds {
2.    struct ipc_perm sem_perm; /* Ownership and
3.      permissions */
4.    time_t sem_otime; /* Last semop time */
5.    time_t sem_ctime; /* Last change time */
6.    unsigned long sem_nsems; /* No. of
7.      semaphores in set */
8.  };
```

semctl()

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, ...);
```

The fourth argument is optional and if present, is of type semun:

```
union semun {
  int val; /* Value for SETVAL */
  struct semid_ds *buf; /* Buffer for IPC_STAT, IPC_SET command*/
  unsigned short *array; /* Array for GETALL, SETALL command*/
  struct seminfo *__buf; /* Buffer for IPC_INF (Linux-specific) */
};
```

semctl() - semid_ds

```
1.  struct semid_ds {
2.    struct ipc_perm sem_perm; /* Ownership and
3.      permissions */
4.    time_t sem_otime; /* Last semop time */
5.    time_t sem_ctime; /* Last change time */
6.    unsigned long sem_nsems; /* No. of
7.      semaphores in set */
8.  };
```

semctl() - cmd

IPC_STAT	Fetch the semid_ds structure for this set, storing it in the structure pointed to by *arg.buf*.
IPC_SET	Set the sem_perm.uid, sem_perm.gid, and sem_perm.modefields from the structure pointed to by *arg.buf* in the semid_ds structure associated with this set. This command can be executed only by a process whose effective user ID equals sem_perm.cuid or sem_perm.uid or by a process with superuser privileges.
IPC_RMID	Remove the semaphore set from the system. This removal is immediate. Any other process still using the semaphore will get an error of EIDRM on its next attempted operation on the semaphore. This command can be executed only by a process whose effective user ID equals sem_perm.cuid or sem_perm.uid or by a process with superuser privileges.
GETVAL	Return the value of semval for the member *semnum*.
SETVAL	Set the value of semval for the member *semnum*. The value is specified by *arg.val*.
GETPID	Return the value of sempid for the member *semnum*.
GETNCNT	Return the value of semncnt for the member *semnum*.
GETZCNT	Return the value of semzcnt for the member *semnum*.
GETALL	Fetch all the semaphore values in the set. These values are stored in the array pointed to by *arg.array*.
SETALL	Set all the semaphore values in the set to the values pointed to by *arg.array*.

semop()

```
1.  #include <sys/types.h>
2.  #include <sys/ipc.h>
3.  #include <sys/sem.h>
4.  int semop(int semid, struct sembuf *sops, size_t nsops);

5.  struct sembuf {
6.    unsigned short sem_num; /* member # in set (0,
7.      1, ..., nsems-1) */
8.    short sem_op; /* operation (negative, 0, or
9.      positive) */
10.   short sem_flg; /* IPC_NOWAIT, SEM_UNDO */
11. };
```

semop()

This value of sem_op can be negative, 0, or positive.

- The easiest case is when sem_op is positive. This case corresponds to the returning of resources by the process.
- If sem_op is negative, we want to obtain resources that the semaphore controls.
- If sem_op is 0, this means that the calling process wants to wait until the semaphore's value becomes 0.

- 下學期OS課會有詳細討論

Semaphore Adjustment on exit

- Whenever we specify the SEM_UNDO flag for a semaphore operation and we allocate resources, the kernel remembers how many resources we allocated from that particular semaphore.
- If an operation specifies SEM_UNDO, it will be automatically undone when the process terminates.

Shared memory

Shared memory

- Fastest form of IPC
  - no need of data copying between client & server
- Must synchronize access to a shared memory segment
  - Semaphores are used
  - Record locking can also be used

functions

- #include <sys/shm.h>
- #include <sys/ipc.h>
- int shmget(key_t key, size_t size, int flag);
- int shmctl(int shmid, int cmd, struct shmid_ds *buf);
- void *shmat(int shmid, const void *addr, int flag);
- int shmdt(void *addr);

Example

```
1.  #include <sys/shm.h>
    #define ARRAY_SIZE 40000
2.  #define MALLOC_SIZE 100000
3.  #define SHM_SIZE 100000
4.  #define SHM_MODE 0600 /* user read/write */
5.  char array[ARRAY_SIZE]; /* uninitialized data = bss */
6.  int main(void) {
7.    int shmid;
8.    char *ptr, *shmptr;
9.    printf("array[] from %lx to %lx\n", (unsigned long)&array[0], (unsigned
    long)&array[ARRAY_SIZE]);
10.   printf("stack around %lx\n", (unsigned long)&shmid);
11.   if ((ptr = malloc(MALLOC_SIZE)) == NULL)
```

Example

```
    err_sys("malloc error");
  printf("malloced from %lx to %lx\n", (unsigned long)ptr, (unsigned long)ptr+MALLOC_SIZE);
  if ((shmid = shmget(IPC_PRIVATE, SHM_SIZE, SHM_MODE)) < 0)
    err_sys("shmget error");
  if ((shmptr = shmat(shmid, 0, 0)) == (void *)-1)
    err_sys("shmat error");
  printf("shared memory attached from %lx to %lx\n",
  (unsigned long)shmptr, (unsigned long)shmptr+SHM_SIZE);
  if (shmctl(shmid, IPC_RMID, 0) < 0)
    err_sys("shmctl error");
}
```

執行結果

```
$ ./tshm
array[] from 0x602100 to 0x60bd40
stack around 0x7ffe5e8eac94
malloced from 0x207c420 to 0x2094ac0
shared memory attached from 0x7f378255a000 to 0x7f37825726a0
```

作業

無