

# Motivation for 'and'

## Synchronization

\*A = 1; \*B = 1;

Process 1

wait(&A);

<use resource A>

signal(&A);

Process 2

wait(&A);

wait(&B);

<use resources A and B>

signal(&B);

signal(&A);

Process 3

wait(&B);

<use resource B>

signal(&B);

# 'and' Synchronization

\*A = 1; \*B = 1;

Process 1

wait(&A);

<use resource A>

signal(&A);

Process 2

wait(&A, &B);

<use resources A and B>

signal(&A, &B);

Process 3

wait(&B);

<use resource B>

signal(&B);

wait(&A, &B) denotes simultaneous wait on A and B. The semaphores A and B are decremented only if both of them can be decremented without blocking

# 'and' Synchronization Implementation

```
wait – if ((ap->value > 0) && (bp->value > 0)) {
    (ap->value)--;
    (bp->value)--;
}
else
    if (ap->value <= 0)
        <add current process to ap->list>
    else
        <add current process to bp->list>
        <block process and reset pc to beginning of wait>

signal – ap->value++;
        <move all processes on ap->list to ready queue>
        bp->value++;
        <move all processes on bp->list to ready queue>
```

Process 1

a1: wait(&A,&B);

b1: <critical section>

c1: signal(&A,&B);

Process 2

a2: wait(&B,&C);

b2: <critical section>

c2: signal(&B,&C);

Process 3

a3: signal(&B,&C);

Assume a1 is executed followed by a2, the semaphore queues are:

A: process 1

B: process 1, process 2

C: process 2

Then, if a3 is executed and the signal only wakes up the first process in each queue, the semaphore queues are:

A: process 1

B: process 2

C:

Now process 1 holds B while blocking on A. Process 2 cannot proceed until process 1 gets A. This defeats the purpose of 'and' synchronization

## Why Wake All

## Processes on Signal?

# POSIX:XSI Semaphores

- POSIX:XSI semaphores are part of the general POSIX:XSI interprocess communication facility
- A POSIX:XSI semaphore is created by executing a `semget` system call
- The `semget` creates a semaphore data structure in the kernel and returns an integer handle to the semaphore
- Processes cannot access semaphore data structures directly – only through system calls
- Semaphore ids or handles are analogous to file descriptors

# POSIX:*SEM* vs POSIX:*XSI* Semaphores

- POSIX:*XSI* data structures are created and kept in the kernel and are referenced through integer handles
- In POSIX:*SEM*, a program declares a variable of type `sem_t` and passes a pointer to that variable

# Semaphore Sets

- A semaphore set is an array of semaphore elements
- A process can perform operations on the entire set in a single system call
- The internal representation of semaphore sets and semaphore elements is not directly accessible
- Each semaphore includes at least the following:
  - A nonnegative integer representing semaphore value
  - Process ID of the last process to manipulate the semaphore element
  - Number of processes waiting for the semaphore element to increase
  - Number of processes waiting for the semaphore element value to equal 0

# Semaphore Sets (Cont)

- Semaphore operations allow a process to block until a semaphore element value is 0 or until it becomes positive
- Each element has two queues associated with it – a queue of processes waiting for the semaphore element value to increase and a queue of processes waiting for the value to equal 0

# Semaphore Creation

## SYNOPSIS

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

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

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

```
int semget (key_t key, int nsems, int  
            semflg);
```

POSIX:XSI/

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#define PERMS S_IRUSR|S_IWUSR
#define SET_SIZE 3
```

```
int semid;
```

```
void main(void)
```

```
{
```

```
    int seimid;
```

```
    if ((semid = semget(IPC_PRIVATE, SET_SIZE, PERMS)) < 0)
        perror("Could not create new private semaphore");
```

```
    else
```

```
        printf("Semaphore created with ID %d\n",semid);
```

```
}
```

# IPC\_PRIVATE

# Random Key

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <string.h>
#include <errno.h>
#define PERMS
S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH
#define SET_SIZE 1
#define KEY ((key_t)99887)
void main(void)
{
    int semid;
    if ((semid = semget(KEY, SET_SIZE, PERMS | IPC_CREAT)) < 0)
        fprintf(stderr, "Error creating semaphore with key %d: %s\n",
            (int)KEY, strerror(errno));
    else
        printf("Semaphore with ID %d created for key
%d\n", semid, (int)KEY);
}
```

# Generate Key from ftok

```
int semid;
key_t mykey;

if (argc != 3) {
    fprintf(stderr, "Usage: %s filename id\n", argv[0]);
    exit(1);
}
if ((mykey = ftok(argv[1], atoi(argv[2]))) == (key_t) -1) {
    fprintf(stderr, "Could not derive key from filename %s: %s\n",
            argv[1], strerror(errno));
    exit(1);
}
else if ((semid = semget(mykey, SET_SIZE, PERMS | IPC_CREAT)) <
0) {
    fprintf(stderr, "Error creating semaphore with key %d: %s\n",
            (int)mykey, strerror(errno));
    exit(1);
}
```

# semop

## SYNOPSIS

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

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

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

```
int semop(int semid, struct sembuf *sops, int nsops);
```

POSIX:XSI

- A process can increment, decrement or test individual semaphore elements with the semop system call
- semid is handle returned by semget
- sops points to an array of element operations
- nsops specifies the number of element operations in the sops array
- semop returns  $-1$  and sets errno on error

# struct sembuf

struct sembuf has the following three fields

- *short sem\_num*: The number of the semaphore element
- *short sem\_op*: The particular operation to be performed on the semaphore element
- *short sem\_flg*: The flags to specify options for the operation

# sem\_op

- If  $\text{sem\_op} > 0$ , semop adds the value to the corresponding semaphore element and awakens processes waiting for semaphore element to increase
- If  $\text{sem\_op} = 0$ , and semaphore element value is not 0, semop blocks the calling process (waiting for 0) and increments the count of processes waiting for a zero value of that element
- If  $\text{sem\_op} < 0$ , semop adds the value to the corresponding semaphore element value provided the result would not be negative. If the operation would make the element value negative, semop blocks the process on the event that the semaphore element value increases. If the resulting value is 0, semop wakes the processes waiting for 0

# set\_sembuf\_struct

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

```
void set_sembuf_struct(struct sembuf *s, int num,
                      int op, int flg)
{
    s->sem_num = (short) num;
    s->sem_op = op;
    s->sem_flg = flg;
    return;
}
```

Do not set sembuf elements in a declaration such as:  
struct sembuf myopbuf = {1, -1, 0}

# Semaphore – Example 1

```
struct sembuf GET_TAPES[2]
struct sembuf RELEASE_TAPES[2]
set_sembuf_struct(&(GET_TAPES[0]), 0,-1,0);
set_sembuf_struct(&(GET_TAPES[1]), 1,-1,0);
set_sembuf_struct(&(RELEASE_TAPES[0]), 0,1,0);
set_sembuf_struct(&(RELEASE_TAPES[1]), 1,1,0);
Process 1:      semop(S, GET_TAPES,1);
                 <use tape A>
                 semop(S, RELEASE_TAPES,1);
Process 2:      semop(S,GET_TAPES,2);
                 <use tapes A and B>
                 semop(S,RELEASE_TAPES,2);
Process 3:      semop(S, GET_TAPES + 1,1);
                 <use tape B>
                 semop(S, RELEASE_TAPES + 1,1);
```

# struct sembuf – Example 1

GET\_TAPES[0]

num = 0

op = -1

flg = 0

RELEASE\_TAPES[0]

num = 0

op = 1

flg = 0

GET\_TAPES[1]

num = 1

op = -1

flg = 0

RELEASE\_TAPES[1]

num = 1

op = 1

flg = 0

# Semaphore – Ex 2 (Top)

```
...
int semid;
int semop_ret;
struct sembuf semwait[1];
struct sembuf semsignal[1];
...
if ( (argc != 2) || ((n = atoi (argv[1])) <= 0) ) {
    fprintf (stderr, "Usage: %s number_of_processes\n", argv[0]);
    exit(1);
}
    /* Create a semaphore containing a single element */
if ((semid = semget(IPC_PRIVATE, SET_SIZE, PERMS)) == -1) {
    fprintf(stderr, "[%ld]: Could not access semaphore: %s\n",
        (long)getpid(), strerror(errno));
    exit(1);
}
```

# Semaphore – Ex 2

## (Upper Middle)

```
/* Initialize the semaphore element to 1 */
set_sembuf_struct(semwait, 0, -1, 0);
set_sembuf_struct(semSignal, 0, 1, 0);

if (semop(semid, semSignal, 1) == -1) {
    fprintf(stderr, "[%ld]: semaphore increment failed - %s\n",
        (long) getpid(), strerror(errno));
    if (remove_semaphore(semid) == -1)
        fprintf(stderr, "[%ld], could not delete semaphore - %s\n",
            (long) getpid(), strerror(errno));
    exit(1);
}
```

# Semaphore – Ex 2

## (Lower Middle)

```
for (i = 1; i < n; ++i)
    if (childpid = fork())
        break;
/* Each child sends pid and ppid info to buffer */
while(( (semop_ret = semop(semid, semwait, 1)) == -1) &&
    (errno == EINTR))
    ;
if (semop_ret == -1)
    fprintf(stderr, "[%ld]: semaphore decrement failed - %s\n",
        (long) getpid(), strerror(errno));
else {
/* Each child prints out its pid and ppid buffer */
while(((semop_ret = semop(semid, semsignal, 1)) == -1) &&
    (errno == EINTR))
    ;
if (semop_ret == -1)
    fprintf(stderr, "[%ld]: semaphore increment failed - %s\n",
        (long) getpid(), strerror(errno));
}
```

# Semaphore – Ex 2 (Bottom)

```
while((wait(&status) == -1) && (errno == EINTR))
    ;
if (i == 1) /* the original process removes the semaphore */
    if (remove_semaphore(semid) == -1)
        fprintf(stderr, "[%ld], could not delete semaphore - %s\n",
            (long)getpid(), strerror(errno));
exit(0);
}
```

# Semaphore – Example 3

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <string.h>
#include <errno.h>
#define SET_SIZE 2
struct sembuf myop[SET_SIZE];
set_sembuf_struct(&(myop[0]), 0, 0, 0);
set_sembuf_struct(&(myop[1]), 0, 1, 0);
if (semop(semid, myop, 2) == -1)
    perror("Semaphore operation failed");
```

# struct sembuf

myop[0]

num = 0

op = 0

flg = 0

myop[1]

num = 0

op = 1

flg = 0

# semctl

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

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

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

```
int semctl(int semid, int semnum, int cmd, /* union semun arg */
```

- semctl queries or sets the values of individual semaphore elements
- semid identifies the semaphore set
- semnum indicates the semaphore element within the set
- cmd refers to individual elements and specifies which command is to be executed
- arg is used differently for different cmd values

# Important Commands

GETVAL:	Return the value of a specific semaphore element
GETPID:	Return process ID of last process to manipulate element
GETNCNT:	Return number of processes waiting for element to increment
GETZCNT:	Return number of processes waiting for element to become 0
SETVAL:	Set value of a specific semaphore element to arg.val
IPC_RMID:	Remove the semaphore identified by semid
IPC_SET:	Set the permissions of the semaphore

semctl returns -1 on error and sets errno – On success the return value is 0 for all commands except GETVAL, GETPID, GETNCNT, and GETZCNT return the value of the command

# semnum Parameter

```
union semnum {  
    int val;  
    struct semid_ds *buf;  
    short *array; };
```

May not be included directly in programs, since some systems do not define it in semaphore header files

```
...
int initialize_sem_element(int semid, int semnum, int semvalue)
{
    union semun arg;
    arg.val = semvalue;
    return semctl(semid, semnum, SETVAL, arg);
}
```

```
void main(void)
```

```
{
    int semid;
    int retval;

    initialize_sem_element

    if ((semid = semget(IPC_PRIVATE, 2, 0600)) < 0)
        perror("Could not create new private semaphore");
    retval = initialize_sem_element(semid, 1, 3);
    printf("initialize_sem_element returned %d\n", retval);
}
```

`initialize_sem_element` returns 0 on success and `-1` and sets `errno` on error

# remove\_semaphore

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

int remove_semaphore(int semid)
{
    return semctl(semid, 0, IPC_RMID);
}
```

# Command Prompt Semaphore Ops

- *List all semaphores:* `ipcs -s`
- *Remove semaphore id 12345:* `ipcrm 12345`

# semop\_restart

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
```

```
int semop_restart(int semid, struct sembuf *sops, int nsops)
{
    int retval;
    while ( ((retval = semop(semid, sops, nsops))
            == -1) &&
            (errno == EINTR) )
        ;
    return retval;
}
```

# sem\_wait\_restart

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

```
#include <errno.h>
```

```
int sem_wait_restart(sem_t *sem)
```

```
{
```

```
    int retval;
```

```
    while ( ((retval = sem_wait(sem)) == -1) &&  
            (errno == EINTR) )
```

```
        ;
```

```
    return retval;
```

```
}
```