Lecture 5-1: IPC(Interproess Communication)

1. Communication between two processes

Accessing a variable or function in the other process is hard. Why?

2. Two approaches: MP, SM

MP(Message Passing): ask OS to transfer the data to the other process

SM(Shared Memory): access the variable in the other process directly with the help of OS

3. messge passing

3.1 Usage

p1:

x = msgget(75, 0777|IPC\_CREAT);

msgsnd(x, &msg, ......);

p2:

x = msgget(75, 0777);

msgrcv(x, &msg);

3.2 data structure

msgget returns "struct msg\_queue".

struct msg\_queue{

 struct kern\_ipc\_perm q\_perm;

 .........

 struct list\_head q\_messages;

};

struct kern\_ipc\_perm{

 .........

 int key;

 unsigned short mode; // permission bit mask

 ........

};

4. Shared memory

4.1) usage

p1:

x = shmget(75, 4, 0777|IPC\_CREAT);

y= (int \*)shmat(x, ......);

p2:

x = shmget(75, 4, 0777);

y = (int \*)shmat(x, ....);

example:

x = shmget(75, 4, 0777|IPC\_CREAT);

if ((fork())==0){ // child

 y= (int \*) shmat(x, 0, 0);

 for(j=0;j<100;j++)

 for(i=0;i<10000;i++)

 \*y = \*y + 1;

 printf("child: the result is %d\n", \*y);

}

else{ // parent

 y= (int \*) shmat(x, 0, 0);

 for(j=0;j<100;j++)

 for(i=0;i<10000;i++)

 \*y = \*y + 1;

 printf("parent: the result is %d\n", \*y);

}

4.2) data structure

shmget() returns "struct shmid\_kernel".

struct shmid\_kernel{

 struct kern\_ipc\_perm shm\_perm;

 struct file \* shm\_file; // this file represents the shared memory

 // this file exists in the physical memory only except

 // during swapped-out state. The physical location of

 // this file is the physical location of the shared memory

 .........

};

shmat() allocates a space in the virtual address space and map this address to the physical location of the shared memory through the page table.

5. Race condition

Shared memory is fast but can cause "race condition". When the outcome of a computation depends on how two or more processes are scheduled, the code is incorrect. We say that there is a race condition.

5.1) Example

The example in 4.1.

5.2) Why it happens?

\*y = \*y + 1 ==>

mov \*y, eax ; \*y -> eax

inc eax

mov eax, \*y

5.3) When it happens?

Critical section: the code area where a shared memory is accessed.

Race condition can happen when two or more processes enter the CS at the same time.

5.4) How to solve?

Basic idea: mutual exclusion.

Mutual exclusion: Make sure only one process enters the CS.

How can we guarantee the mutual exclusion?

method:

 All processes check before entering CS if there is another process in the CS.

 If there is one

 wait until it comes out

 Else

 enter

5.5) Implementing mutual exclusion

without os, pl

 sw : dekker, peterson

 hw : cli/sti, tsi

with os

 semaphore

with pl

 monitor

5.6) sw

P1, P2:

lp1: mov r0, lock

 cmp r0, 1

 je lp

 mov lock, 1

 -- CS --

 mov lock, 0

- lock=0 initially

- enter if lock=0

5.7) hw

P1,P2:

 cli

 --CS--

 sti

Or

 lp: tsl ro, lock

 cmp r0, 1

 je lp

 --CS--

 mov lock, 0

tsl ro, lock is same as the following operations all executed atomically

 ro = lock

 if (r0==0)

 lock=1

5.8) semaphore

P1,P2:

wait(s);

--CS--

signal(s);

wait(s):

 s = s - 1;

 if (s < 0)

 someone is already in CS. wait here. (insert into waiting list on s)

 else

 enter CS

signal(s):

 s = s + 1;

 if (s <=0)

 someone is in the waiting list. wake one up.

5.9) using semaphore

int semid;

struct sembuf psembuf={0,-1,SEM\_UNDO};

struct sembuf vsembuf={0,1,SEM\_UNDO};

main(){

.............

semid = semget(75, 1, 0777|IPC\_CREAT); // get a semaphore

sem\_union.val=1;

semctl(semid, 0, SETVAL, sem\_union); // initial value is 1

semop(semid, &psembuf, 1); // wait(s)

--CS--

semop(semid, &vsembuf, 1); // signal(s)

5.10) data structure for semaphore

semget() returns "struct sem\_array".

struct sem\_array{

 struct kern\_ipc\_perm sem\_perm;

 struct sem \* sem\_base; // link list of sem structure

 .........

};

struct sem{

 unsigned long semval; // value of this semaphore

 ......

};

5.11) Deadlock

Using semaphore correctly not easy. Example: producer-consumer problem.

[producer] produce item and push to the stk

lp: get\_item(&item);

if (top==MAX)

 sleep();

top=top+1;

stk[top]=item

if (top==1)

 wakeup(소비자);

goto lp;

[consumer] pop item from stk and consume

lp: if (top==0)

 sleep();

item = stk[top];

top = top-1;

if (top==MAX-1)

 wakeup(생산자);

consume(item);

goto lp;

mutex : intialize with 1. sema for mutual exclusion

holes : initial val=MAX. sema for holes

items : initial val=0. sema for items produced so far

[producer]

lp: get\_item(&item);

wait(holes);

wait(mutex);

top=top+1;

stk[top]=item

signal(mutex);

signal(items);

goto lp;

[consumer]

lp:

wait(items);

wait(mutex);

item = stk[top];

top = top-1;

signal(mutex);

signal(holes);

consume(item);

goto lp;

6. Homework

1) Try below (ex1.c) and explain the result.

#include <stdio.h>

unsigned long long sum=0;

void main(){

int x=fork();

 if (x==0){ // child

 int i, j;

 for(i=0;i<20000;i++)

for(j=0;j<2000;j++)

sum++;

 printf(“child sum:%llu\n”,sum);

 }else{ // parent

int i, j;

 for(i=0;i<20000;i++)

for(j=0;j<2000;j++)

sum++;

 printf(“parent sum:%llu\n”,sum);

 }

}

#gcc -o ex1 ex1.c

#./ex1

2) Try below (th.c) and explain the result.

#include <stdio.h>

#include <string.h>

#include <pthread.h>

#include <stdlib.h>

#include <unistd.h>

pthread\_t t1, t2; // thread 1, thread 2

unsigned long long sum=0;

void \* foo1(void \*arg){

 int i,j;

 for(i=0;i<20000;i++){

 for(j=0;j<2000;j++)

 sum += 1;

 }

 printf(“thread 1 sum:%llu\n”, sum);

 return NULL;

}

void \* foo2(void \*arg){

 int i,j;

 for(i=0;i<20000;i++){

 for(j=0;j<2000;j++)

 sum += 1;

 }

 printf(“thread 2 sum:%llu\n”, sum);

 return NULL;

}

int main(void){

pthread\_create(&t1, NULL, &foo1, NULL);

pthread\_create(&t2, NULL, &foo2, NULL);

pthread\_join(t1, NULL);

 pthread\_join(t2, NULL);

return 0;

}

# gcc -o th -lpthread th.c

#./th

....

#./th

....

#./th

....

3. Try below(th2.c) and explain the result.

#include <stdio.h>

#include <string.h>

#include <pthread.h>

#include <stdlib.h>

#include <unistd.h>

pthread\_t t1, t2; // thread 1, thread 2

pthread\_mutex\_t lock; // semaphore

unsigned long long sum=0;

void \* foo1(void \*arg){

 int i,j;

 for(i=0;i<20000;i++){

 pthread\_mutex\_lock(&lock);

 for(j=0;j<2000;j++)

 sum += 1;

 pthread\_mutex\_unlock(&lock);

 }

 printf(“thread 1 sum:%llu\n”, sum);

 return NULL;

}

void \* foo2(void \*arg){

 int i,j;

 for(i=0;i<20000;i++){

pthread\_mutex\_lock(&lock);

 for(j=0;j<2000;j++)

 sum += 1;

pthread\_mutex\_unlock(&lock);

 }

 printf(“thread 2 sum:%llu\n”, sum);

 return NULL;

}

int main(void){

 pthread\_mutex\_init(&lock, NULL);

pthread\_create(&t1, NULL, &foo1, NULL);

pthread\_create(&t2, NULL, &foo2, NULL);

pthread\_join(t1, NULL);

 pthread\_join(t2, NULL);

 pthread\_mutex\_destroy(&lock);

return 0;

}

# gcc -o th2 -lpthread th2.c

#./th2

...

#./th2

...

#./th2

.....

4. (Deadlock) Try below(th3.c) and explain the result. Modify the code so that it won't have a deadlock.

#include <stdio.h>

#include <string.h>

#include <pthread.h>

#include <stdlib.h>

#include <unistd.h>

pthread\_t t1, t2; // thread 1, thread 2

pthread\_mutex\_t lock1; // semaphore 1 for sum 1

pthread\_mutex\_t lock2; // semaphore 2 for sum 2

unsigned long long sum1=0;

unsigned long long sum2=0;

void \* foo1(void \*arg){

 int i,j;

 for(i=0;i<20000;i++){

pthread\_mutex\_lock(&lock1);

pthread\_mutex\_lock(&lock2);

 for(j=0;j<2000;j++)

 sum1 += 1;

pthread\_mutex\_unlock(&lock1);

for(j=0;j<2000;j++)

 sum2 += 1;

pthread\_mutex\_unlock(&lock2);

 }

 printf(“thread 1 sum1:%llu\n”, sum1);

printf(“thread 1 sum2:%llu\n”, sum2);

 return NULL;

}

void \* foo2(void \*arg){

 int i,j;

 for(i=0;i<20000;i++){

pthread\_mutex\_lock(&lock2);

pthread\_mutex\_lock(&lock1);

 for(j=0;j<2000;j++)

 sum1 += 1;

pthread\_mutex\_unlock(&lock1);

for(j=0;j<2000;j++)

 sum2 += 1;

 pthread\_mutex\_unlock(&lock2);

 }

 printf(“thread 2 sum1:%llu\n”, sum1);

printf(“thread 2 sum2:%llu\n”, sum2);

 return NULL;

}

int main(void){

pthread\_mutex\_init(&lock1, NULL);

 pthread\_mutex\_init(&lock2, NULL);

pthread\_create(&t1, NULL, &foo1, NULL);

pthread\_create(&t2, NULL, &foo2, NULL);

pthread\_join(t1, NULL);

 pthread\_join(t2, NULL);

 pthread\_mutex\_destroy(&lock1);

pthread\_mutex\_destroy(&lock2);

return 0;

}

# gcc -o th3 -lpthread th3.c

#./th3

...

#./th3

...

#./th3

.....

5. Find out the ISR2 function for pthread\_mutex\_lock() and trace the code. You can do kernel tracing also in the following site: <https://elixir.bootlin.com/linux/latest/ident/>. Select the right version (v2.6.25.10) and type the ISR2 name in the search box.