|  |
| --- |
| **국민대학교 컴퓨터공학부** |
| Operating System |
| **Homework #4 Dining Philosopher** |

|  |
| --- |
| 20093267 김 성 근  2014-04-28 |

**1. Dining Philosopher 문제를 Semaphore를 이용하여 구현하시오.**

•젓가락 한 짝은 Semaphore 주머니의 구슬 한 개로 표현.

•Deadlock에 대한 대비가 없이 작성.

•printf()를 적절히 사용하여 중간 상태를 출력.

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#define SEMPERM 0600

#define TRUE 1

#define FALSE 0

typedef struct union\_semun {

int val;

struct semid\_ds \*buf;

ushort \*array;

} semun;

int initsem (key\_t semkey, int n){

int status = 0, semid;

if ( (semid = semget (semkey, 1, SEMPERM | IPC\_CREAT | IPC\_EXCL) ) == -1){

if ( (errno == EEXIST) )

semid = semget (semkey, 1, 0);

}else{

semun arg;

arg.val = n;

status = semctl(semid, 0, SETVAL, arg);

}

if (semid == -1 || status == -1){

perror("initsem failed");

return (-1);

}

return (semid);

}

int p (int semid){

struct sembuf p\_buf;

p\_buf.sem\_num = 0;

p\_buf.sem\_op = -1;

p\_buf.sem\_flg = SEM\_UNDO;

if ( (semop(semid, &p\_buf, 1) ) == -1){

perror ("p(semid) failed");

exit(1);

}

return (0);

}

int v (int semid){

struct sembuf v\_buf;

v\_buf.sem\_num = 0;

v\_buf.sem\_op = 1;

v\_buf.sem\_flg = SEM\_UNDO;

if ( (semop(semid, &v\_buf, 1)) == -1){

perror ("v(semid) failed");

exit(1);

}

return (0);

}

void dining(int semid1, int semid2, int chopstick1, int chopstick2){

int status;

semun arg;

pid\_t pid = getpid();

status = semctl(semid1, 0, GETVAL, arg);

printf("\nnumber of sempaphore ID[ %d ] : %d\n", semid1, status);

status = semctl(semid2, 0, GETVAL, arg);

printf("number of sempaphore ID[ %d ] : %d\n", semid2, status);

p(semid1);

printf("process %d Hold Chop Stick [ %d ] \n", pid, chopstick1);

printf("process %d Wait Chop Stick [ %d ] : [ %d ]\n", pid, chopstick1, chopstick2);

sleep(5);

p(semid2);

printf("process %d Hold Chop Stick [ %d ] \n", pid, chopstick2);

printf("process %d Eat Chop Stick [ %d ] : [ %d ]\n", pid, chopstick1, chopstick2);

v(semid1);

v(semid2);

printf("process %d Pull Chop Stick [ %d ] : [ %d ]\n", pid, chopstick1, chopstick2);

exit(0);

}

int main(){

key\_t semkey[] = { 0x200, 0x300, 0x400, 0x500, 0x600 };

int semid[5];

int i;

for( i = 0; i < 5 ; i++){

if ( (semid[i] = initsem(semkey[i],1) ) < 0)

exit(1);

}

for( i = 0; i < 5; i++){

if(fork() == 0){

if( i == 4){

dining(semid[4], semid[0], 4, 0);

}

dining(semid[i], semid[i+1], i, i+1);

}

}

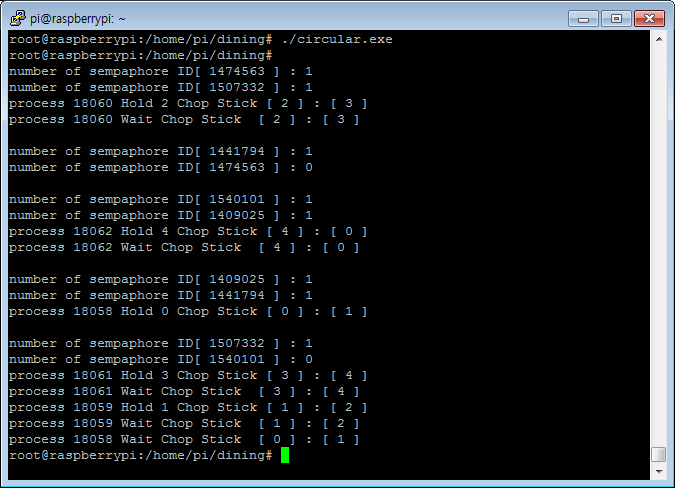
return 0;

}

Code 1. circular.c

위 함수는 Homework 3(Synchronization by Semaphore)에서 했던 코드를 기반으로 하였습니다. Semaphore를 이용하여 젓가락을 구현하기 위해서 semkey와 semid를 배열로 각각 5개씩 주었습니다. 이후 dining 함수에서 p()를 연속으로 두 번 실행하도록 구현하고 이 때에 Hold & Wait를 구현하기 위하여 p() 사이에 sleep(time) 함수를 이용 parameter time 초 동안 sleep 되고 그 다음이 실행되도록 구현 하였습니다.

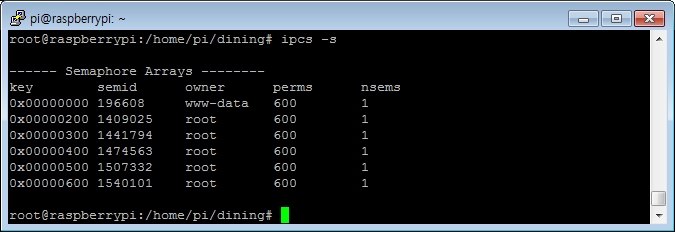
그리고 저희가 배운 Semaphore에서는 저희가 현재 지금 가지고 있는 Semaphore의 개수를 볼 수 없지만 현재 Linux의 Advanced IPC에서 제공하는 함수를 이용하여 Semaphore가 있는지 없는지를 확인하였습니다.



Screen 1. Dining Philosopher DeadLock 실행화면

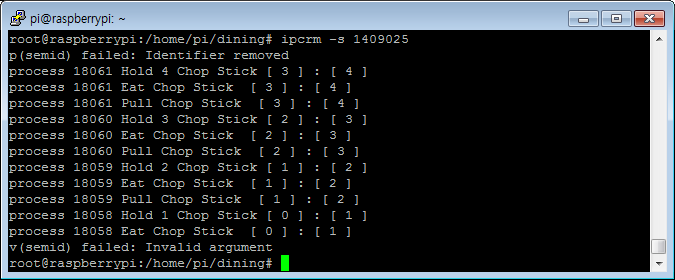
위 화면에서 볼 수 있듯이 처음에 젓가락 [2,3]을 가지려는 사람이 2젓가락을 가져가고 그 다음 젓가락 [4,0]을 가지려고 하는 사람이 4젓가락을 가져가고 젓가락[0,1]을 가져가려는 사람이 0을 가져가고 젓가락 [3,4]을 가려가라는 사람이 3을 가져가고 그 다음 젓가락 [1,2]를 가져가려는 사람이 젓가락 1을 가져가는 바람에 Deadlock 이 걸린 상황입니다.

[0,1]=0, [1,2]=1, [2,3]=3, [3,4]=3, [4,0]=4 로 젓가락을 각자가 하나씩 가져갔기 때문에 Deadlock이 걸렸고 이후에 아무런 반응이 존재하지 않습니다.



Screen 2. Linux Semaphore 확인 명령어

위에 Linux에서 제공되는 명령어 “ ipcs –s “를 이용하여 실행을 확인해본 결과 semaphore가 존재하는 것을 확인 할 수 있으며 제거하는 명령어 “ ipcrm –s ID “를 실행시켜보면 아래와 같이 deadlock이 걸린 상태가 해제됨을 알 수 있습니다. semid가 하나 제거 되었기 때문에 p(semid) failed : Indentifier removed / v(semid) failed : Invalid argument 가 뜨고 나머지 Process는 정상적으로 실행되었음을 확인 할 수 있습니다.



Screen 3. Linux Semaphore 제거 화면

**2. Deadlock을 Circular wait 조건을 깨는 방법을 이용하여 Prevention(예방)**

int main(){

key\_t semkey[] = { 0x200, 0x300, 0x400, 0x500, 0x600 };

int semid[5];

int i;

for( i = 0; i < 5 ; i++){

if ( (semid[i] = initsem(semkey[i],1) ) < 0)

exit(1);

}

for( i = 0; i < 5; i++){

if(fork() == 0){

**if( i == 4){**

**dining(semid[0], semid[4], 0, 4);**

**}**

dining(semid[i], semid[i+1], i, i+1);

}

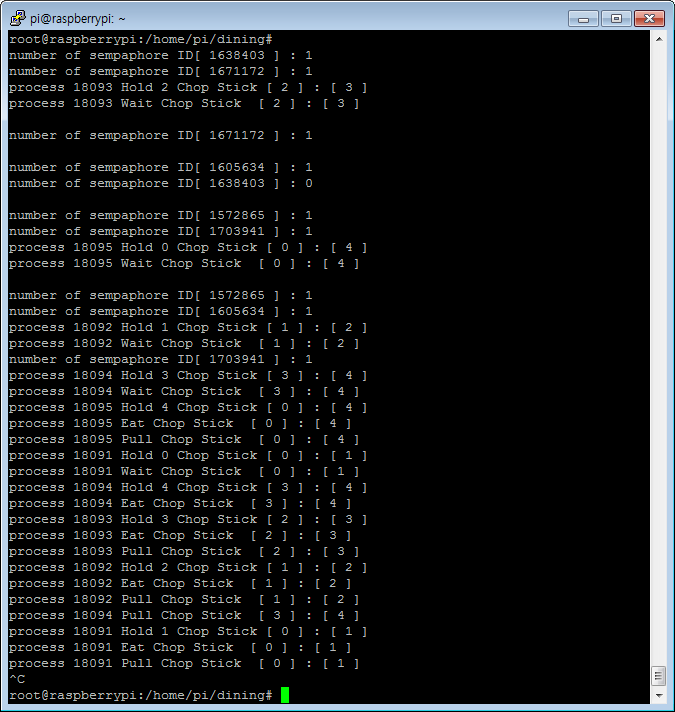
}

return 0;

}

Code 2. Ciruclar Wait Prevention code

Deadlock을 예방하기 위해서 Circular wait 조건을 깨는 방법을 이용하여 Prevention(예방)한 코드입니다. 위의 코드와 다르게 순서를 바꾼 것입니다. 기존에 [0,1], [1,2], [2,3], [3,4], [4,0] 이였던 방식을 [0,1], [1,2], [2,3], [3,4], **[0,4]** 로 바꾸어 Deadlock이 걸리지 않도록 수행한 것 입니다. 수행 결과는 아래와 화면과 같습니다. 흰색으로 밑줄 쳐준 부분을 통해 5번다 실행되었음을 확인할 수 있습니다.



Screen 4. Circular Wait Prevention 실행화면

**3. Deadlock을 Hold & Wait조건을 깨는 방법을 이용하여 Prevention(예방)**

void dining(key\_t mutexKey, int semid1, int semid2, int chopstick1, int chopstick2){

int status;

int mutexId;

semun arg;

pid\_t pid = getpid();

if ( (mutexId = initsem(mutexKey,1) ) < 0)

exit(1);

printf("\nprocess %d before critical section\n", pid);

p(mutexId);

printf("process %d in critical section\n",pid);

// ~ Critical Section Start

status = semctl(semid1, 0, GETVAL, arg);

printf("\nnumber of sempaphore ID[ %d ] : %d\n", semid1, status);

status = semctl(semid2, 0, GETVAL, arg);

printf("number of sempaphore ID[ %d ] : %d\n", semid2, status);

p(semid1);

printf("process %d Hold %d Chop Stick [ %d ] : [ %d ] \n", pid, chopstick1, chopstick1, chopstick2);

printf("process %d Wait Chop Stick [ %d ] : [ %d ]\n", pid, chopstick1, chopstick2);

sleep(5);

p(semid2);

printf("process %d Hold %d Chop Stick [ %d ] : [ %d ]\n", pid, chopstick2, chopstick1, chopstick2);

printf("process %d Eat Chop Stick [ %d ] : [ %d ]\n", pid, chopstick1, chopstick2);

v(semid1);

v(semid2);

printf("process %d Pull Chop Stick [ %d ] : [ %d ]\n", pid, chopstick1, chopstick2);

// ~ Critical Section End

printf("process %d leaving critical section\n", pid);

v(mutexId);

printf("process %d exiting\n",pid);

exit(0);

}

int main(){

key\_t mutexKey= 0x100;

key\_t semkey[] = { 0x200, 0x300, 0x400, 0x500, 0x600 };

int semid[5];

int i;

for( i = 0; i < 5 ; i++){

if ( (semid[i] = initsem(semkey[i],1) ) < 0)

exit(1);

}

for( i = 0; i < 5; i++){

if(fork() == 0){

if( i == 4){

dining(mutexKey,semid[4], semid[0], 4, 0);

}

dining(mutexKey,semid[i], semid[i+1], i, i+1);

}

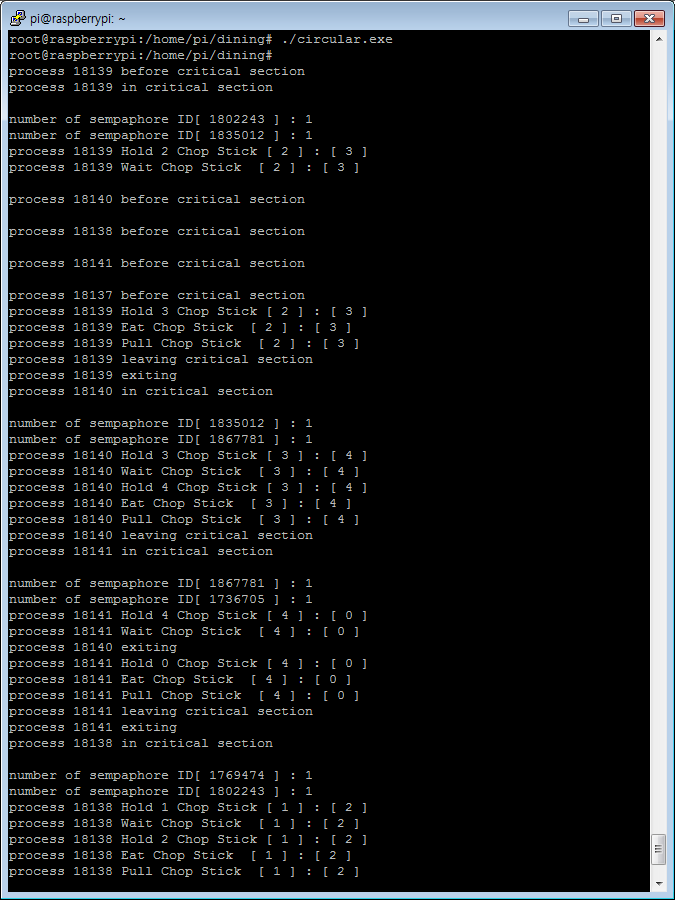
}

return 0;

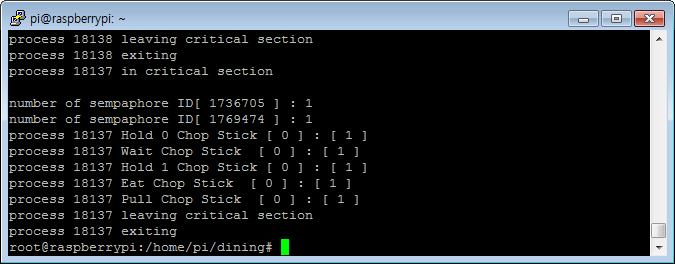
}

Code 3. Hold & Wait Prevention Code

Hold & Wait를 막기 위해서 Mutex Semaphore을 이용 Lock의 기능을 사용하여 한번에 모두 실행하도록 하여 조건을 깨는 방법 입니다. 위의 코드를 작성한 것과 같이 Semaphore를 하나 더 사용한다. 실행화면은 아래와 같습니다. 흰색으로 밑줄 쳐준 부분을 통해 5번다 실행되었음을 확인할 수 있습니다.



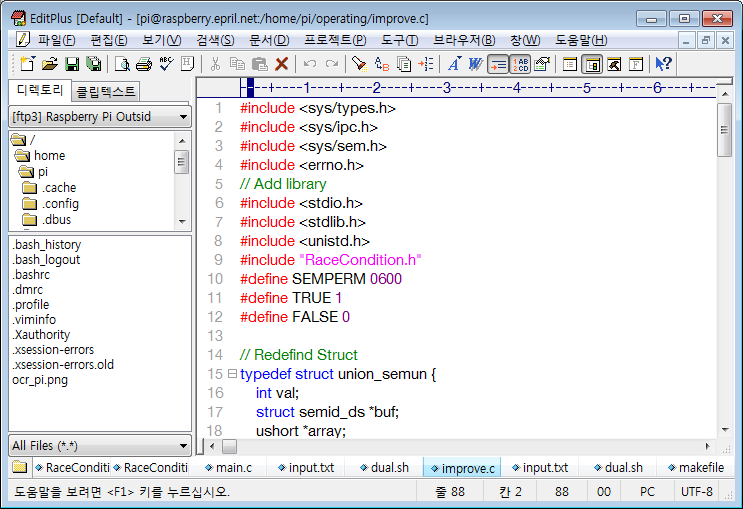
Screen 5. Hold & Wait Prevention 실행화면



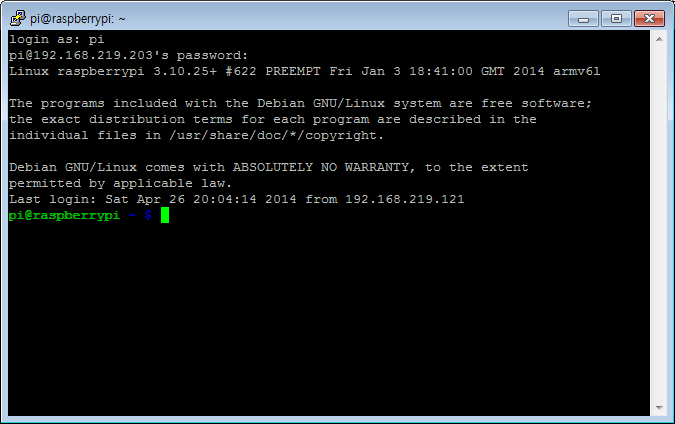
Screen 6. Hold & Wait Prevention 실행화면 (cont)

**3. 기타 : 개발환경, 실행화면**

개발환경은 RaspberryPi의 Raspbian Linux 운영체제를 이용하였고 EditPlus 프로그램을 이용하여 소스코드를 수정하였으며, Putty를 이용 SSH접근을 통하여 실행하였습니다.



Screen 7. Editplus 실행화면



Screen 8. Putty 실행화면