CECS 326

Spring 2017

Haney Williams

Project 5

Phanna Chrin

Amanda Pan

**Given Problem:**

Write a program using C/C++ to do the following:

Create N child processes. Each child process writes its child number, its PID, its parent PID, and its child

PID in a buffer. Create a critical section where each child process displays the content of the buffer. Create a semaphore mechanism to protect the critical section. Use a delay adjustment parameter k in the critical section to adjust the speed of the display process to show that without semaphore protection the displayed contents of the buffer are randomly interleaved.

semaphore N opt k

where:

N: number of child processes (integer)

opt: option: “n” -> no semaphore protection, “s” -> with semaphore protection

k: delay adjustment parameter (integer)

**Problem Analysis:**

This project is to understand the functionality of semaphores. The program takes in arguments of the *filename* command, the number of processes to be forked, semaphore option, and the delay speed assigned to each semaphore. The program then generates a unique id via *ftok()* for the semaphore identifier, sets the semaphore to 1 (semaphore is created regardless) and forks the number of processes. If “s” is chosen as the option, the semaphore waits for a process to be finished before decrementing itself to 0 and locks the current process. The output is put into a char buffer and *fputc()* is called to read the char values until the pointer points to null. If “s” is chosen, the semaphore increments itself to signal other processes that it is done. The parent process waits for the child process to be done before killing itself and removes the semaphores.

**Program Flowchart:**

User Inputs arguments

Create semaphore

Fork process

If option is “s”, lock process

Else, do nothing

Put output into a buffer

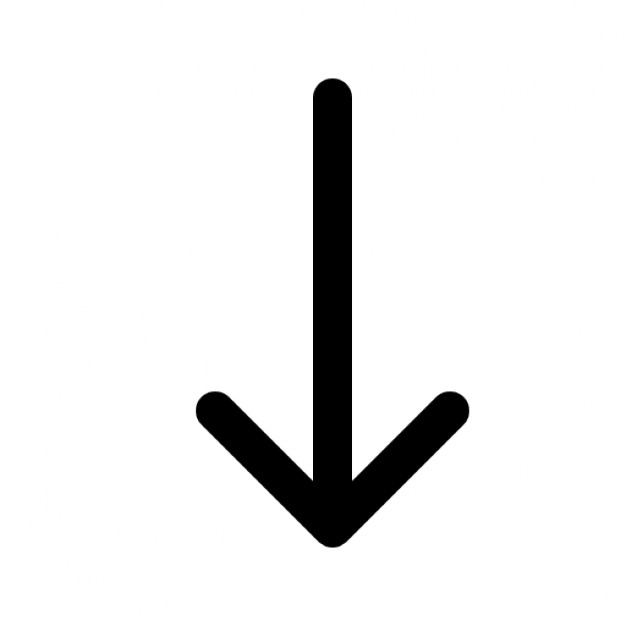
Set buffer to stream mode

If option is “s”, unlock process

Else do nothing

Parent process kills itself when child process is done

Parent process removes the semaphore set



**C Code:**

#include <stdio.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <unistd.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <sys/wait.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;

}

int main (int argc, char \*argv[]) {

int N, k, opt, i, status;

pid\_t childPid, pid = getpid();

key\_t ipc\_key;

int semid, semop\_ret, sem\_value;

char buf\_arr[8192], \*c\_ptr;

    struct semid\_ds sem\_buf;

/\* check for invalidation \*/

if (argc != 4) {

printf("Need: ./filename #process semaphoreOption(n/s) #delay\n");

exit(1);

}

N = atoi(argv[1]);

k = atoi(argv[3]);

if (N < 0 || k < 0) {

printf("Number of processes and delay adjustment speed must be positive integers\n");

exit(1);

}

if (strcmp(argv[2],"n") != 0 && strcmp(argv[2],"s") != 0) {

        printf("The semaphore argument must be either 'n' or 's' only\n");

        exit(1);

    }

/\* semaphore option \*/

    if (strcmp(argv[2], "n") == 0) opt = 0; // no semaphore protection

            if (strcmp(argv[2], "s") == 0) opt = 1; // Semaphore protection

/\* declare semaphore wait and increment \*/

struct sembuf semwait[1];

struct sembuf semsignal[1];

/\* initialize semaphore element to 1 \*/

set\_sembuf\_struct(semwait, 0, -1, 0);

set\_sembuf\_struct(semsignal, 0, 1, 0);

ipc\_key = ftok(".", 'S'); // Generate a key from a pathname

/\* create semaphore \*/

    /\* semaphore is always created regardless of semaphore procetion. It is only used if k = s \*/

    if ((semid = semget(ipc\_key, 1, IPC\_CREAT | IPC\_EXCL | 0666)) == -1) {

perror ("semget: IPC | 0666");

exit(1);

    }

/\* increment semaphore \*/

if (semop(semid, semsignal, 1) == -1) {

printf ("%ld: semaphore increment failed - %s\n", (long)getpid(), strerror(errno));

if (semctl(semid, 0, IPC\_RMID) == -1) // Remove the semaphore if unable to increment

printf ("%ld: could not delete semaphore - %s\n", (long)getpid(), strerror(errno));

exit(1);

}

/\* fork N processes \*/

for(i = 1; i < N; i++) {

if((childPid = fork()) == -1) { //fails

perror("fork failed\n");

exit(1);

}

if(childPid > 0) break;//parent process

}

if(opt == 1) {

while (((semop\_ret = semop(semid, semwait, 1)) == -1) && (errno == EINTR));

if (semop\_ret == -1)

printf ("%ld: semaphore decrement failed - %s\n", (long)getpid(), strerror(errno));

}

/\* create the output message and put in into char buffer \*/

sprintf(buf\_arr,"i: %d: process ID: %6ld parent ID: %6ld child ID: %6ld\n",(i),(long)getpid(), (long)getppid(), (long)childPid);

c\_ptr = buf\_arr; // pointer points to the buffer

/\* if buffer is a null pointer, buffering is disabled for the stream, which becomes an unbuffered stream \*/

setbuf(stdout, NULL); // set the buffer to stream for I/O operations

/\* go through the char buffer using the pointer until it points to NULL \*/

while (\*c\_ptr != '\0') {

fputc(\*c\_ptr, stderr);

usleep(k); // sleep in usec microsecond using the delay adjustment parameter

c\_ptr++;

}

if (opt == 1) {

while (((semop\_ret = semop(semid, semsignal, 1)) == -1) && (errno == EINTR));

if (semop\_ret == -1)

printf ("%ld: semaphore increment failed - %s\n", (long)getpid(), strerror(errno));

}

waitpid(childPid, &status, 0); // wait for the child process to complete before the parent is killed

/\* if this is the parent process, remove the semaphore set \*/

if(pid == getpid()) {

if(semctl(semid, 0, IPC\_RMID) == -1) {

printf("%ld: couldn't delete semaphore - %s\n", (long)getpid(), strerror(errno));

exit(1);

}

}

exit(0);

}

**Output:**

