Deadlock Avoidance

Joseph Carrillo

Operating Systems Fundamentals

CST-321

6/5/22

Flow Chart Screen Shot

Diagram

Description automatically generated

Loom Link Video of .c Code and Command Line

<https://loom.com/share/b86cca352ea5402097d636c74a7e6a94>

DeadLock Avoidance Command Line Output

Text

Description automatically generated

Encountered Segmentation Error on Command Line

A screenshot of a computer

Description automatically generated

.c Source Code

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

#include <signal.h>

#include <semaphore.h>

#include <pthread.h>

#include <sys/mman.h>

// global variables

int sem\_post(sem\_t \*sem);

int sem\_trywait(sem\_t \*sem);

int sem\_wait(sem\_t \*sem);

sem\_t\* semaphore;

pid\_t otherPid;

sigset\_t sigSet;

void signalHandlerOne(int signum){

printf("Caught Signal: %d\n",signum);

printf("Exit Child Process: \n");

sem\_post(semaphore);

\_exit(0);

}

void signalHandler2(int signum){

printf("I am alive!\n");

}

// logic to run to simulation a parent Process

void childProcess(){

// set the signalHandlers

signal(SIGUSR1, signalHandlerOne);

signal(SIGUSR2, signalHandler2);

// child process: simulates a hung processs waiting for a semaphore pr

// sem\_postout if its running too long

int value;

sem\_getvalue(semaphore,&value);

printf("Child process semaphore count is: %d.\n",value);

printf("Child Process is grabbing semaphore.\n");

sem\_wait(semaphore); // semaphore lock on semaphore

sem\_getvalue(semaphore,&value);

printf("Child Process semaphore count is: %d \n",value);

//beginning of critical region// to where more than one processes access the same code

printf("Starting very long child process.. \n");

for(int x =0; x < 60; ++x){

printf(".\n"); // wait time

}

// end critical region

printf("Exit Child process: \n");

\_exit(0);

}

void \*checkHungChild(void \*a){

// simulate a timer of 10 seconds by going to sleep then checki if semaphore is locked indicating a hung

int\* status = a;

printf("Check for a hung childProcess..\n");

sleep(10);// wait ten seconds

// if else statement

if(sem\_trywait(semaphore) != 0){ // is locked by sem

printf("child proces hung.\n");

\*status = 1; // if sem is greater than 0

} else{

printf("child process is running.\n"); // if sem is = o

\*status = 0;

}

return NULL;

}

//logic to run a simulate a parent process

void parentProcess(){

// detect hung child process and kill it after timing out

sleep(2); // wait two seconds

// process id is returned

if(getpgid(otherPid) >=0){

printf("ChildProcess is running.\n"); // output childprocess running

}

int value;

sem\_getvalue(semaphore,&value);

printf("Inside the parent process. \n semaphore count: %d.\n",value);

// try to get semaphore & if it is locked start timer

if(sem\_trywait(semaphore) !=0){

// Timer thread is started and waits for it to return

pthread\_t tid1; // tid1

int status =0;

printf("Child is hung &/or running too long: ...\n"); // output

// if statement

if(pthread\_create(&tid1, NULL, checkHungChild, &status)) // new thread is created to check attributes

printf("Error: Timer Thread.\n");

\_exit(1);

}

if(pthread\_join(tid1,NULL)){

printf("\n Error: Joining timer therad.\n");

\_exit(1);

}

// kill child processif(status)

if(status ==1){

// kill child processif

printf("Child process kill: \n ID of: %d\n",otherPid);

// kill

(otherPid,SIGUSR1);

kill(otherPid,SIGTERM);

printf("(Killed Child Process: \n");

}

// prove that child process is killed

printf("Proven, Child process Killed. \n");

sleep(5); // wait five seconds

kill(otherPid,SIGUSR2); // kill signal to "otherPid"

sleep(1); // wait one second

printf("Done, Child process killed.\n"); // out completed process

// try to get semaphore

sem\_getvalue(semaphore,&value);

printf("In the parent process, Semaphore count: %d.\n",value);

// if semaphore is not locked

if(sem\_trywait(semaphore)!=0){

if(value ==0) // if value is greater than one, semaphore is locked

sem\_post(semaphore);

printf("Successly got semaphore.\n"); // output success of retrieving semaphore

sem\_getvalue(semaphore,&value);

printf("In parent process, Semaphore count: %d.\n",value);

}else{

printf("Completed semaphore.\n");

}

}

// Exit parent process

printf("Exit Parent Process.\n");

\_exit(0);

}

// main

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

pid\_t pid;

//create shared semaphore

if(sem\_init(semaphore, 1,1) !=0){

printf("Failed to create semaphore.\n");

exit(EXIT\_FAILURE);

}

// use fork()

// the output from both the child and the parent process should be written

// to both run at the same timer

// pid = fork();

if(pid == -1 )

{

// error if fork() returns -1

printf("Fork, Error:\n");

exit(EXIT\_FAILURE);

}

// fork() returns 0 then success

if(pid ==0){

// run child process logic

printf(" Start Child Processs.\n Process ID: %d\n",getpid());

otherPid = getppid();

childProcess();

} else{

// Running parent process

printf("Sart Parent Process.\n Process ID: %d\n",getpid());

otherPid = pid;

parentProcess();

}

// close/killsem\_destroy(semaphore);

//

sem\_destroy(semaphore);

return 0;

}