Anthony Redamonti

Syracuse university

Lab 3

CIS-657 Principles of operating systems

prof. mohammed abdallah

1/28/2021

Part 1: Xsh\_create.c

The following xsh\_create.c program creates a shell command that creates processes. The argument to the command is the priority of the process and is optional. If no priority is provided, then the default value of the priority if 20. The runforever process is then created and resumed.

/\* xsh\_create.c - xsh\_create \*/

#include <xinu.h>

#include <stdio.h>

#include <string.h>

/\*------------------------------------------------------------------------

 \* xsh\_create - shell command to create a process

 \*------------------------------------------------------------------------

 \*/

shellcmd xsh\_create(int nargs, char \*args[])

{

    pid32   pid;            /\* return PID           \*/

    pri16   priority;       /\* process priority > 0     \*/

    char    ch;             /\* next character of argument   \*/

    char    \*chptr;         /\* walks along argument string  \*/

    /\* Output info for '--help' argument \*/

    if (nargs == 2 && strncmp(args[1], "--help", 7) == 0) {

        printf("Usage: %s Priority\n\n", args[0]);

        printf("Description:\n");

        printf("\tcreates a process\n");

        printf("Options:\n");

        printf("\tPriority \tthe priority of a process to create\n");

        printf("\t\t If no priority provided, a default value\n");

        printf("\t\t of 20 is used\n");

        printf("\t--help\tdisplay this help and exit\n");

        return OK;

    }

    /\* Check argument count \*/

    if (nargs > 2) {

        fprintf(stderr, "%s: incorrect argument\n", args[0]);

        fprintf(stderr, "Try '%s --help' for more information\n", args[0]);

        return SYSERR;

    }

/\* compute priority from argument string \*/

    if (nargs == 2) {

        chptr = args[1];

        ch = \*chptr++;

        priority = 0;

        while(ch != NULLCH) {

            if ( (ch < '0') || (ch > '9') ) {

                fprintf(stderr, "%s: non-digit in process ID\n",

                    args[0]);

                return 1;

            }

            priority = 10\*priority + (ch - '0');

            ch = \*chptr++;

        }

        pid = create(runforever, 1024, priority, "p 1", 0);

        if (pid == SYSERR) {

            fprintf(stderr, "%s: cannot create process %d\n",

                args[0], priority);

            return 1;

        }

        resume(pid);

    }

    /\* No argument so use priority default value of 20 \*/

    if (nargs == 1){

        pid = create(runforever, 1024, 20, "p 1", 0);

        if (pid == SYSERR) {

            fprintf(stderr, "%s: cannot create process %d\n",

                args[0], priority);

            return 1;

        }

        resume(pid);

    }

    return 0;

}

The runforever process is defined in runforever.c shown below. It will print out the current PID and then call an infinite while-loop.

/\* runforever.c - runforever \*/

#include <xinu.h>

/\*------------------------------------------------------------------------

 \*  runforever  -  run forever in an infinite while-loop

 \*------------------------------------------------------------------------

 \*/

void runforever(void)

{

    pid32 pid = getpid();

    printf("PID of created process is: %d\n", pid);

    while(1);

}

1. Asdf
2. If the infinite while-loop is removed from the runforever process, the process terminates

Part 2: Using a Semaphore

Below is the program written to create two processes “m1” and “m2”. Both processes share a semaphore “sem” to control access to an array. M1 will generate numbers 1-15 each having an entry in the array. M1 will then block while M2 prints the contents of the array. M2 will signal the process waiting on the semaphore (M1) to produce 15 more numbers starting at the first index. The process continues until the numbers 1-1995 have been printed to the console. To ensure that m1 runs first, it is given a priority of 40, and m2 is given a priority of 20. The output of the code is on the next page.

/\* main.c - main \*/

#include <xinu.h>

void m1();void m2();

sid32 sem;

pid32 m1pid, m2pid;

int32 array[15];

void main(void)

{

    sem = semcreate(15);

    m1pid=create(m1, 1024, 40,"m1",0);

    m2pid=create(m2, 1024, 20,"m2",0);

    resume(m1pid);

    resume(m2pid);

    return OK;

}

void m1(){

    int32 i;

    int32 count = 0;

    for (i = 1; i <= 2000; i++) {

        wait(sem);

        array[count] = i;

        if(++count == 15){

            count = 0;

        }

    }

    kill(m2pid);

}

void m2(){

    int32 j;

    while(1) {

        for(j = 0; j < 15; j++){

            kprintf("array[%d] = %d\n", j, array[j]);

        }

        signaln(sem,15);

    }

}



