**COSC430 Quiz #1**

Create a C program with the following process synchronization behavior. This synchronization process continues forever. You need to provide your own signal handling functions in the Parent, Child1, Child2, and Child3, each printing out a message with the cumulative number of signals received. You may use the “synchro.c” program as a template code given in p. 144 of the textbook.

Parent Process

(6) SIGUSR1

(1) SIGUSR1

(5) SIGUSR1

(4) SIGUSR1

(3) SIGUSR1

(2) SIGUSR1

Child 2

Child 3

Child 1

Programing Code:

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <unistd.h>

#include <sys/wait.h>

volatile sig\_atomic\_t counter[4] = {0}; // array to hold signal counts

void parent\_handler(int sig) {

counter[0]++;

printf("Parent received %d signals\n", counter[0]);

}

void child1\_handler(int sig) {

counter[1]++;

printf("Child 1 received %d signals\n", counter[1]);

}

void child2\_handler(int sig) {

counter[2]++;

printf("Child 2 received %d signals\n", counter[2]);

}

void child3\_handler(int sig) {

counter[3]++;

printf("Child 3 received %d signals\n", counter[3]);

}

int main() {

pid\_t child1, child2, child3;

sigset\_t mask;

struct sigaction act;

// set up signal handling for parent process

act.sa\_handler = parent\_handler;

sigemptyset(&act.sa\_mask);

act.sa\_flags = 0;

if (sigaction(SIGUSR1, &act, NULL) == -1) {

perror("sigaction");

exit(EXIT\_FAILURE);

}

// set up signal handling for child 1

act.sa\_handler = child1\_handler;

sigemptyset(&act.sa\_mask);

act.sa\_flags = 0;

if (sigaction(SIGUSR1, &act, NULL) == -1) {

perror("sigaction");

exit(EXIT\_FAILURE);

}

// set up signal handling for child 2

act.sa\_handler = child2\_handler;

sigemptyset(&act.sa\_mask);

act.sa\_flags = 0;

if (sigaction(SIGUSR1, &act, NULL) == -1) {

perror("sigaction");

exit(EXIT\_FAILURE);

}

// set up signal handling for child 3

act.sa\_handler = child3\_handler;

sigemptyset(&act.sa\_mask);

act.sa\_flags = 0;

if (sigaction(SIGUSR1, &act, NULL) == -1) {

perror("sigaction");

exit(EXIT\_FAILURE);

}

// block SIGUSR1 in the parent process

sigemptyset(&mask);

sigaddset(&mask, SIGUSR1);

if (sigprocmask(SIG\_BLOCK, &mask, NULL) == -1) {

perror("sigprocmask");

exit(EXIT\_FAILURE);

}

// create child processes

if ((child1 = fork()) == -1) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (child1 == 0) { // child 1 process

while (1) {

kill(getppid(), SIGUSR1);

sleep(1);

}

} else if ((child2 = fork()) == -1) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (child2 == 0) { // child 2 process

while (1) {

kill(getppid(), SIGUSR1);

sleep(2);

}

} else if ((child3 = fork()) == -1) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (child3 == 0) { // child 3 process

while (1) {

kill(getppid(), SIGUSR1);

sleep(3);

}

}

// unblock SIGUSR1 in the parent process

if (sigprocmask(SIG\_UNBLOCK, &mask, NULL) == -1) {

perror("sigprocmask");

exit(EXIT\_FAILURE);

}

// wait for child processes to finish

if (waitpid(child1, NULL, 0) == -1) {

perror("waitpid");

exit(EXIT\_FAILURE);

}

if (waitpid(child2, NULL, 0) == -1) {

perror("waitpid");

exit(EXIT\_FAILURE);

}

if (waitpid(child3, NULL, 0) == -1) {

perror("waitpid");

exit(EXIT\_FAILURE);

}

// print final signal counts

printf("Parent received %d signals\n", counter[0]);

printf("Child 1 received %d signals\n", counter[1]);

printf("Child 2 received %d signals\n", counter[2]);

printf("Child 3 received %d signals\n", counter[3]);

return 0;

}

Output:

A picture containing text, font, screenshot, typography

Description automatically generated