**Operating Systems 2012-2013**

**Final examination**

1. **(20 points)** Write a C program that searches an array of integers for another given integer. To speedup the search, the search has to be done in parallel by two child processes. The parent process reads in the number of integers and then the integers in an array. It also reads in the integer to be searched. It then creates two child processes. The first child process searches the first half of the array, and the second child process searches the second half. If the integer is found, its index in the array is sent to the parent through a pipe. If it is not found, a -1 is sent to the parent through a pipe. The parent waits for both child processes to finish and then prints an appropriate message.

Solution:

#include<stdlib.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/ipc.h>

#include <string.h>

void main ()

{

int i,status,num1,num2,num3,num4, fd1[2], fd2[2];

int a[1000];

char b[5],c[5],d[5],e[5];

pid\_t pid1,pid2;

printf("\n\n\nEnter how many numbers:");

scanf("%d",&num1);

printf("\n\nEnter the %d numbers:\n",num1);

for (i=0;i<num1;i++)

{

printf("%d : ",i);

scanf("%d",&a[i]);

}

printf("\n\nEnter the number to search:");

scanf("%d",&num2);

pipe(fd1);

pipe(fd2);

pid1=fork();

if (pid1==0)

{

printf("this is the child 1\n");

for (i=0;i<(num1/2);i++)

{

if (a[i]==num2)

{

printf("found by process 1\n");

sprintf(b,"%d",i);

sprintf(c,"%d",-1);

write(fd1[1],&b,4);

write(fd2[1],&c,4);

break;

}

printf("%d\n",a[i]);

}

\_exit ( EXIT\_FAILURE ) ;

}

else

if (pid1>0)

{

pid2=fork();

if (pid2==0)

{

printf("this is the child 2\n");

for (i=(num1/2);i<num1;i++)

{

if (a[i]==num2)

{

printf("found by process 2\n");

sprintf(b,"%d",-1);

sprintf(c,"%d",i);

write(fd1[1],&b,4);

write(fd2[1],&c,4);

break;

}

printf("%d\n",a[i]);

}

\_exit(EXIT\_FAILURE);

}

}

if (waitpid (pid1, &status, 0)>0 && waitpid (pid2, &status, 0)>0)

{

read(fd1[0],d,4);

read(fd2[0],e,4);

num3=atoi(d);

num4=atoi(e);

if (num3>0) printf("value of i is %d\n",num3);

if (num4>0) printf("value of i is %d\n",num4);

}

}

**II. (30 points).** A disk is formatted with a file system for UNIX with a block size of 4KB and has a block address size of 4Bytes and a traditional inode structure (10 direct pointers, 1 simple indirect pointer, 1 double indirect pointers, and 1 triple indirect pointers). Answer the following questions:

1. How many disk blocks require the following files (including data and metadata blocks):

* File A having 20 KBytes
* File B having 200 KBytes
* File C having 2000 KBytes
* File D having 20000 KBytes

1. What is the maximum size of a file in this file system?
2. What is the maximum number of files that this file system can have? Justify your answer.

**Solution:**

Each block fits 1024 disk addresses: 4KBytes /4 bytes =1024.

1. A : 5 blocks + 1 block for the inode: 6

B: 50 blocks + 1 block of simple indirect addresses + 1block for the inode = 52

C: 500 bloques de datos+ 1 block of simple indirect addresses + 1block for the inode = 502.

D: 5000 blocks + 1 block of simple indirect addresses + 5 blocks of simple indirect addresses + 1block for the inode = 5007

1. The maximum number of blocks:
   * 10 direct
   * 1024 simple indirect
   * 1024\*1024 double indirect
   * 1024\*1024\*1024 triple indirect

Aprox: 4TBytes

1. Limited by the number of inodes, but the size of an inode is not indicated in this problem, neither the space reserved for the inodes. It cannot be told.

**III. (20 points)** A new entertaining and multi-adventure center has open in Leganés. Multiple players can use the center at the same time and can get involved in a number of playing activities. For making sure that all installations work properly, a maintenance team has to perform a number of tasks including cleaning, changing lights, check the state of the equipment, etc.

You are asked to write a program simulating the working of this center in C pseudo-code. For synchronization you can use either ***semaphores*** or ***mutexes and condition variables*** for synchronization. Your program must:

* Implement a function player() to code the behavior of the players.
* Implement a function worker() to code the behavior of the workers.
* Create 100 player threads
* Create 5 maintenance threads
* The players can simultaneously use the equipment, but they cannot start to use it if any maintenance worker is currently working
* The maintenance workers can enter the center only when there is no player inside. Additionally, given that they work in turns, there will be a maximum of one worker in the center at any given time.
* The thread player thread checks that there is no worker inside the center. When a player enters the center, he blocks the worker access (if necessary), prints its player number on the screen, simulates the utilization of the installations by randomly waiting between 1 and 5 seconds, and finishes execution. The last player must release the access to the workers.
* The worker thread blocks the access of the players, prints his worker number (pthread\_self()) on the screen, and simulates the maintenance by randomly waiting between 1 and 3 seconds, and prints a message that the work is done.

Below you can see a possible output of the program:

player 1 enters the game

player 2 enters the game

player 2 leaves the center

player 1 leaves the center

worker 2236799856 working

worker 2236799856 finishes work

player 3 enters the game

player 4 enters the game

player 4 leaves the center

player 5 enters the game

player 3 leaves the center

player 5 leaves the center

worker 2220014448 working

worker 2220014448 finishes work

.

.

**Solution:**

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <semaphore.h>

#include <pthread.h>

#include <stdlib.h>

int n\_players = 0; /\* number of players \*/

sem\_t sem\_player;

sem\_t mutex; /\* controls the access to the park \*/

void \*player(void) { int pl;

sleep (random()%5);

sem\_wait(&sem\_player);

n\_players = n\_players + 1;

pl=n\_players;

if (n\_players == 1) sem\_wait(&mutex);

sem\_post(&sem\_player);

printf("player %d enters the game \n", jug);

sem\_wait(&sem\_player);

n\_players = n\_players - 1;

if (n\_players == 0) sem\_post(&mutex);

sem\_post(&sem\_player);

printf("player %d leaves the center \n", pl);

pthread\_exit(0);

}

void \*worker() {

sem\_wait(&mutex);

printf("worker %u working\n", (unsigned int)pthread\_self());

sleep(random()%3);

printf("worker %u finishes work\n", (unsigned int) pthread\_self());

sem\_post(&mutex);

pthread\_exit(0);

}

void main(void) {

pthread\_t th[105];

int i;

sem\_init(&mutex, 0, 1);

sem\_init(&sem\_player, 0, 1);

for (i=0;i<100;i++)

pthread\_create(&th[i], NULL, player, NULL);

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

pthread\_create(&th[i+100], NULL, worker, NULL);

for (i=0;i<105;i++)

pthread\_join(th[i], NULL);

exit(0);

}

**IV.** **(10 points)** Quiz questions (***NOTE: A correct answer scores 1point, a wrong answer subtracts 0.25 points, an unanswered question scores 0 points***)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **Answer** | **D** | **B** | **B** | **C** | **B** | **D** | **B** | **D** | **B** | **C** |

1. Race condition means: a) When one process executes a piece of code, it “excludes” all the others from executing at the same time. b) When one process executes a piece of code, it “excludes” all the other from executing it at any time in the future c) Several processes executing at the same time target to be as fast as possible. d) Several processes access some data concurrently and the outcome depends on the access order
2. Let S be a semaphore initialized to 1, and P0 a process running **signal(S); ---; wait(S);** and P1 a process running **signal(S);---;wait(S);** The above situation may cause: a) Starvation b) A race condition c) Deadlock d) No problem
3. Segmentation is a memory management scheme that: a) Organizes the memory in fixed-size segments b) Organizes the memory in logical segments c) Is implemented in software in the OS.
4. A bit map can be used for a) disk space allocation b) free space management c) both of the above d) None of the above
5. Each file has an associated counter, which counts the number of symbolic links pointing to the file. a) True. b) False .
6. What is false about a system call? a) It can be implemented through a trap b) It has a regular library interface. c) Changes the execution to system mode d) Causes always a context switch
7. Processes P1, P2, and P3 with execution times 15, 30, and 20ms arrive in this order. If the short-time round-robin scheduler has a quantum of 10ms, what time does P1 finish? a) 60 b) 35 c) 50 d) 65
8. The short-time scheduler selects a process from a) Already created processes before entering the ready queue b) Blocked processes c) Swapped processes d) Ready queue
9. A short quantum of round robin scheduler has a negative impact on: a) Fairness b) CPU usage efficiency c) Response time for interactive users d) Difficulty of implementing the algorithm
10. A path containing the root directory is called a) Active b) Passive c) Absolute d) Relative

**V. (20 points)** Give definitions of the following concepts in maximum 3 lines:

1. UNIX consistency
2. Deadlock
3. Condition variable
4. Fragmentation

Solution:

Slides see Silberschatz book