**Exercise 2.**

**Answer Sheet**

Student's Name and ID: **Yuta Nemoto s1240234**

***Problem 1.*** (15 points)Consider the following program:

#include <stdio.h>

main () {

long int i;

long int n = 500000000;

while (1) {

/\* ~ 1 second run \*/

printf ("\*Ready\*\n");

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

/\* 2 seconds sleep \*/

printf ("\*Sleep\*\n");

sleep (2);

}

}

a) Save this source code in a file called loop.c, compile it, and run it.

b) Answer to the following questions.

1. Using command "top", check the status of the process your program has created. Checking should be performed every second. Which command options can be used? (Consult the manual page for “top”) (Zero: it doesn't allow to check status every second)

The command option is “-pid”

I can check the state of process by using command “top -pid [PID of process]”

In my case, I used the command “top -pid 10363” and see the changing of status like below.

Processes: 294 total, 2 running, 1 stuck, 291 sleeping, 994 threads 11:18:38

Load Avg: 1.57, 1.70, 1.71 CPU usage: 2.46% user, 1.97% sys, 95.56% idle SharedLibs: 242M resident, 43M data, 61M linkedit. MemRegions: 25929 total, 2041M resident, 104M private, 495M shared.

PhysMem: 8073M used (1562M wired), 115M unused. VM: 782G vsize, 627M framework vsize, 0(0) swapins, 0(0) swapouts. Networks: packets: 26131519/29G in, 15217632/3646M out. Disks: 7826449/58G read, 5259953/116G written.

PID COMMAND %CPU TIME #TH #WQ #POR MEM PURG CMPR PGRP PPID STATE BOOSTS %CPU\_ME %CPU\_OTHRS UID FAUL COW MSGS MSGR SYSB SYSM CSW PAGE IDLE POWE USER #MRE RPRV VPRV VSIZ KPRV KSHR

10877 a.out 0.0 00:11.60 1 0 10 292K 0B 0B 10877 9871 sleeping \*0[1] 0.00000 0.00000 17518 440 42 36 18 102 44 2943 0 7 0.0 s1240234 N/A N/A N/A N/A N/A N/A

1. Show the states of the process created by your program as seen from the top command output in the table below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | Time, (sec) | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Running | \* |  |  | \* |  |  | \* | \* |  |
| Sleeping |  | \* | \* |  | \* | \* |  |  | \* |

(put \* in the appropriate cells)

**Problem 2.** (15 points)Consider the following program:

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

int main () {

pid\_t pid, pid1;

/\* fork a child process \*/

pid = fork();

if (pid < 0) { /\* error occured \*/

fprintf (stderr, "Fork failed");

return 1;

}

else if (pid == 0) { /\* child process \*/

pid1 = getpid();

printf ("child: pid = %d\n", pid); /\* A \*/

printf ("child: pid1 = %d\n", pid1); /\* B \*/

}

else { /\* parent process \*/

pid1 = getpid();

printf ("parent: pid = %d\n", pid); /\* C \*/

printf ("parent: pid1 = %d\n", pid1); /\* D \*/

wait (NULL);

}

return 0;

}

What will be printed out at lines A, B, C, and D assuming that the actual **pid**s of the parent and the child are 4500 and 4505, respectively?

/\* A \*/ 0

/\* B \*/ 4505

/\* C \*/ 4505

/\* D \*/ 4500

***Problem 3.***. (20 points) Consider a process P, which needs to have 2 CPU-time units, 3 time units for I/O, and again 2 CPU-time units for its execution.

**P:**

|  |  |  |
| --- | --- | --- |
| CPU-time (2units) | I/O time (3units) | CPU-time (2units) |

NOTES: Process should be in the ready queue at least one time unit before leaving the ready state. The time needed to create the process takes one time unit.

Continue the following time-state diagram:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State |  | | | | | | | | | | | | | |
| Running |  |  | \* |  | \* |  |  |  |  | \* |  | \* |  |  |
| Ready |  | \* |  | \* |  |  |  |  | \* |  | \* |  |  |  |
| Waiting |  |  |  |  |  | \* | \* | \* |  |  |  |  |  |  |
| New | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Terminated |  |  |  |  |  |  |  |  |  |  |  |  | \* |  |
| Time units → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |

***Problem 4.*** (30 points)Consider the process P from problem 3. Let it produce the following child process CH during its first time unit of execution:

CH:

|  |  |  |
| --- | --- | --- |
| CPU-time (1units) | I/O time (2units) | CPU-time (2units) |

Make time-state diagram for the processes P and CH for the following cases: (Attention: In both cases, child process appear at state new at time 3, after parent process switch to running state.)

**Case 1**. Parent process P is waiting until the child-process CH will be finished. If both processes are staying in the Ready state, then the child will be chosen for execution.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | PROCESS P | | | | | | | | | | | | | | | | | | | | | |
| Running |  |  | \* |  |  |  |  |  |  |  |  |  | \* |  |  |  |  | \* |  | \* |  |  |
| Ready |  | \* |  |  |  |  |  |  |  |  |  | \* |  |  |  |  | \* |  | \* |  |  |  |
| Waiting |  |  |  | \* | \* | \* | \* | \* | \* | \* | \* |  |  | \* | \* | \* |  |  |  |  |  |  |
| New | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Terminated |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | \* |  |
| Time → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | PROCESS CH | | | | | | | | | | | | | | | | | | | | | |
| Running |  |  |  |  |  | \* |  |  |  | \* |  | \* |  |  |  |  |  |  |  |  |  |  |
| Ready |  |  |  |  | \* |  |  |  | \* |  | \* |  |  |  |  |  |  |  |  |  |  |  |
| Waiting |  |  |  |  |  |  | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| New |  |  |  | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Terminated |  |  |  |  |  |  |  |  |  |  |  |  | \* |  |  |  |  |  |  |  |  |  |
| Time → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |

**Case 2**. Parent process P is executing concurrently with the child-process CH. Please, take into account that the child process has higher priority.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | PROCESS P | | | | | | | | | | | | | | | | | | | | | |
| Running |  |  | \* |  | \* |  |  |  |  |  | \* |  | \* |  |  |  |  |  |  |  |  |  |
| Ready |  | \* |  | \* |  |  |  |  | \* | \* |  | \* |  |  |  |  |  |  |  |  |  |  |
| Waiting |  |  |  |  |  | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| New | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Terminated |  |  |  |  |  |  |  |  |  |  |  |  |  | \* |  |  |  |  |  |  |  |  |
| Time → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | PROCESS CH | | | | | | | | | | | | | | | | | | | | | |
| Running |  |  |  |  |  | \* |  |  |  | \* |  | \* |  |  |  |  |  |  |  |  |  |  |
| Ready |  |  |  |  | \* |  |  |  | \* |  | \* |  |  |  |  |  |  |  |  |  |  |  |
| Waiting |  |  |  |  |  |  | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| New |  |  |  | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Terminated |  |  |  |  |  |  |  |  |  |  |  |  | \* |  |  |  |  |  |  |  |  |  |
| Time → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |

***Problem 5.*** (20 points) Answer the following questions

1. Explain the differences between a process and thread.

The main differences are

* + 1. The flow of processing
    2. Sharing of resource

First, process take their own memory space and it needs to create child process to execute the same program at the same time. On the other hand, thread shares almost all the data with other threads, so mainly it doesn't take their own memory space, and it's processed in parallel processing normally. Second, process share the resource only with child process, so one process can't access to memory of the other process directly. On the other hand, thread share the resource with the other threads. Sometimes it occurs some trouble in using variable shared with other threads.

1. Consider a situation when the parent-process is terminated before than its dependent child-process was finished. What strategy can you suggest to resolve this situation? Please take into account that the child process can also have its own child processes.

As one of the strategy, that process can continue their own processing by re-parented. In Unix-oriented system, normally child process get 'init' process as a parent process. For other ways, I have some options like exterminating the child process or setting a timer to end the expiration of child process.