CIS 520 - Operating Systems I – Homework #1 Solution

Due: Monday, Sept. 23rd, by 11:59 pm, upload via K-State OnLine or turn in on paper.

1. Consider the code shown in the textbook in Figure 4.13 (also in the public directory as /pub/CIS520/programs/prog4.13.c):

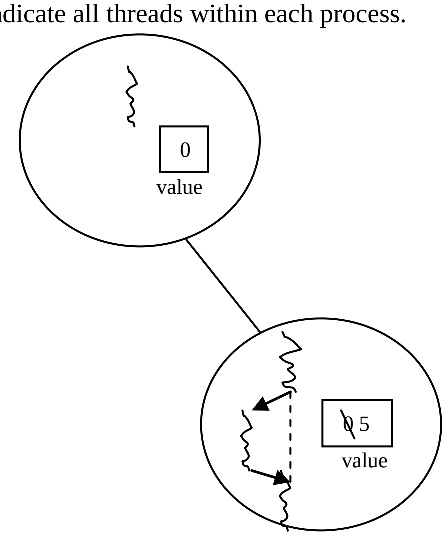
```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param);

int main(int argc, char *argv[])
{
  int pid;
  pthread_t tid;
  pthread_attr_t attr;

  pid = fork();

  if (pid == 0) {
    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,runner,NULL);
    pthread_join(tid,NULL);
    printf("CHILD: value = %d\n", value);  /* LINE C */
  }
  else if (pid > 0) {
    wait(NULL);
    printf("PARENT: value = %d\n", value); /* LINE P */
  }
}

void *runner(void *param) {
  value = value + 5;
  pthread_exit(0);
}
```

Compile using the command: **`gcc –o prog4 prog4.13.c –lpthread`**.

a. Upon execution, what would be the output from the program at **`LINE C`** and **`LINE P`**?

CHILD: value = 5
PARENT: value = 0

b. Draw the Process Model, clearly indicate all instances of the variable **value** and all changes that are made to it. Also, clearly indicate all threads within each process.



If you just have two squiggly lines inside of the child process to indicate two threads, that is sufficient.

c. Suppose that the two lines that are used to create and join with a new thread are duplicated; that is, what would be output if

```
pthread_create(&tid,&attr,runner,NULL);
pthread_join(tid,NULL);
```

becomes

```
pthread_create(&tid,&attr,runner,NULL);
pthread_join(tid,NULL);
pthread_create(&tid,&attr,runner,NULL);
pthread_join(tid,NULL);
```

How would the process model change?

Two child threads are created in the child process, and the second one also increments the value by 5, so the output would be:

CHILD: value = 10
PARENT: value = 0

d. What if those two lines become

```
pthread_create(&tid,&attr,runner,NULL);
pthread_create(&tid,&attr,runner,NULL);
pthread_join(tid,NULL);
pthread_join(tid,NULL);
```

What would be output, and how would the process model change? Is the runner function thread-safe? Is the output always going to be the same?

The model is essentially the same, but the runner function is NOT thread-safe. Different outputs are possible, including the two given above with:

CHILD: value = 10
PARENT: value = 0

Or

CHILD: value = 5
PARENT: value = 0

The first output is likely to occur most often because it doesn't take very long to inc value by 5.

e. Finally, suppose that we replace the `runner()` function in the previous part 1d with:

```
void *runner(void *param) {
  int i;
  for (i=1; i<=100000; i++)
    value = value + 5;
  pthread_exit(0);
}
```

Explain the output observed. In particular, why are different outputs observed on subsequent runs?

The code is NOT thread-safe, so different interleavings of the threads result in different output. If all updates are made correctly, then each child should add 5 * 100,000 = 500,000, so the result would be:

CHILD: value = 1000000
PARENT: value = 0

If some updates are lost, then the result will be less.

2. Including the initial parent process, how many processes are created when the following program is executed? Draw the Process Model showing the parent-child hierarchy.

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    /* fork off a child process */
    fork();

    /* fork off another child process */
    fork();

    /* and another */
    fork();

    /* wait for a signal to be received */
    pause();
}
```

Hint: you can use the pstree command to generate the process tree, and then kill all of the processes in the group identified by the initial parent's id (pid); e.g., **`kill -9 -<pid>`**.

A total of $2^3 = 8$ process are created in an asymmetric tree. Here is the output shown using:
$ ps --forest -u neilsen (for user "neilsen"):

```
18769 pts/13  00:00:00      \_ bash
21793 pts/13  00:00:00          \_ procs
21794 pts/13  00:00:00          |   \_ procs
21797 pts/13  00:00:00          |   |   \_ procs
21800 pts/13  00:00:00          |   |   |   \_ procs
21799 pts/13  00:00:00          |   |   \_ procs
21795 pts/13  00:00:00          |   \_ procs
21798 pts/13  00:00:00          |   |   \_ procs
21796 pts/13  00:00:00          |   \_ procs
21824 pts/13  00:00:00          \_ ps
```

$ kill -9 -21793

A traditional process model with a tree rooted by a single node, with three children, two grandchildren, and one great grandchild is fine as well.

3. Suppose that ***k*** **`fork`**`()` statements are executed in order before pausing as shown above in problem 2 with ***k*=*3***. How many processes would be created for any ***given k*** > ***0***?

   Hint: Write a formula as a function of ***k***.

   Number of processes created = 2^k

4. Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use nonpreemptive scheduling and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time = Run Time
P1	0.0	8.0
P2	0.4	4.0
P3	1.0	1.0

   a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

   Average turnaround time = ((8-0) + (12-0.4) + (13-1))/3 = (8+11.6+12)/3 = 31.6/3 = 10.5333..

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

Average turnaround time = ((8-0) + (9-1) + (13-0.4))/3 = (8+8+12.6)/3 = 9.5333..

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process P1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit of time and then SJF scheduling is used. Remember that processes P1 and P2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge or crystal ball scheduling.

Average turnaround time = ((2-1)+ (6-0.4) + (14-0))/3 = (1+5.6+14)/3 = 6.866..