Chapters3and4

February 24, 2023 11:18 AM

Review Questions – Processes & Threads and Concurrency (Chapters 3 and 4)

Operating Systems SFWRENG 3SH3 Term 2, Winter 2023
Prof. Neerja Mhaskar

Q1) Using the program below explain what the output will be at LINE A and why?

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int value = 5;

int main()
{
pid_t pid;

  pid = fork();

  if (pid == 0) { /* child process */
    value += 15;
    return 0;
  }
  else if (pid > 0) { /* parent process */
    wait(NULL);
    printf("PARENT: value = %d",value); /* LINE A */
    return 0;
  }
}
```

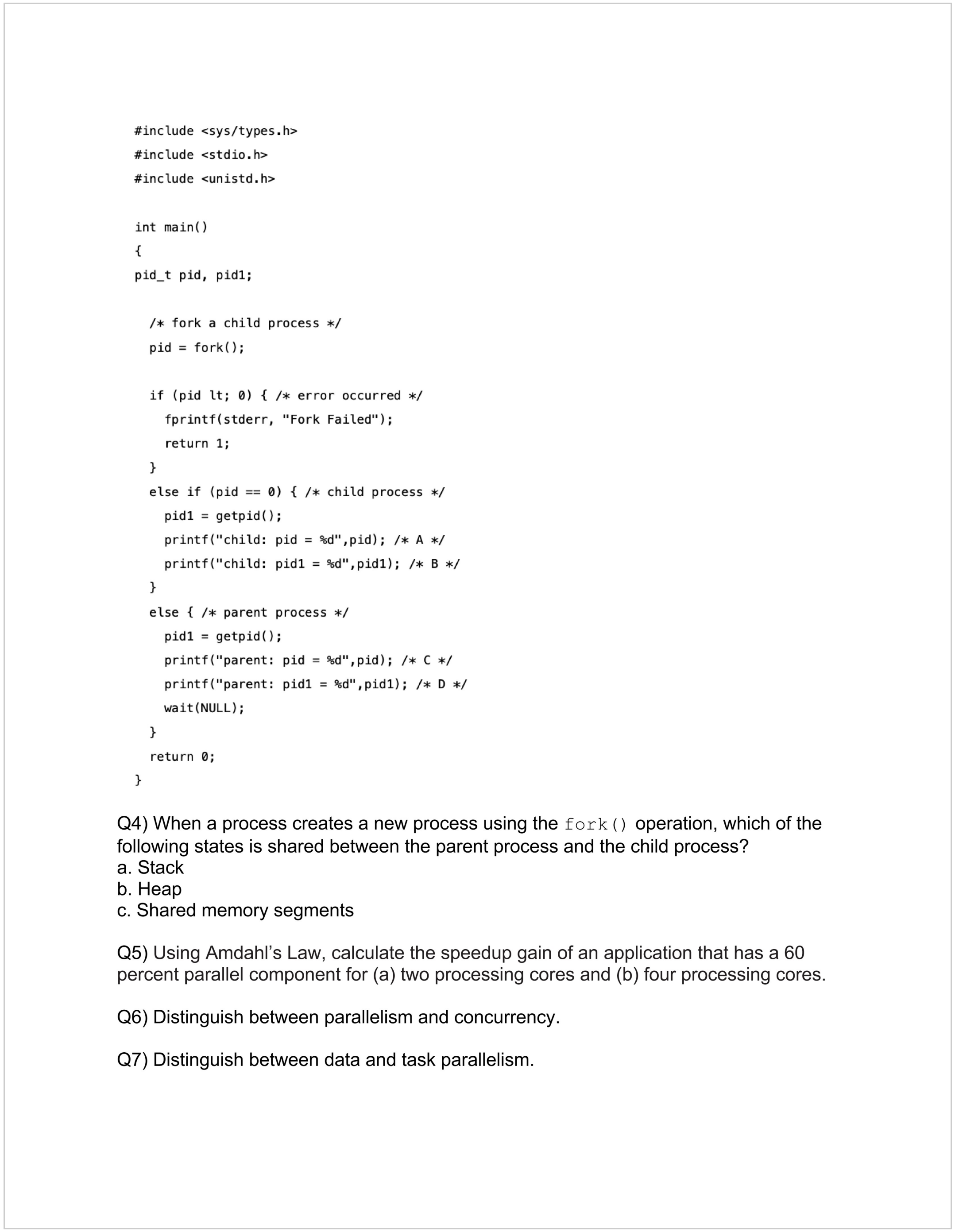
1) The output would be 5, since value is only incremented inside the child process

Q2) Including the initial parent process, how many processes are created by the program shown below? Construct a tree of processes as explained in class for the processes created.

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

{
int main()
{
  int i;

{
  for (i = 0; i < 4; i++)
   fork();
{

  return 0;
}
```

Q3) Using the program below, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

2) 2^4 = 16



```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid, pid1;

  /* fork a child process */
  pid = fork();

  if (pid lt; 0) { /* error occurred */
    fprintf(stderr, "Fork Failed");
    return 1;
  }
  else if (pid == 0) { /* child process */
    pid1 = getpid();
    printf("child: pid = %d",pid); /* A */
    printf("child: pid1 = %d",pid1); /* B */
  }
  else { /* parent process */
    pid1 = getpid();
    printf("parent: pid = %d",pid); /* C */
    printf("parent: pid1 = %d",pid1); /* D */
    wait(NULL);
  }
  return 0;
}
```

Q4) When a process creates a new process using the `fork()` operation, which of the following states is shared between the parent process and the child process?
a. Stack
b. Heap
c. Shared memory segments

Q5) Using Amdahl's Law, calculate the speedup gain of an application that has a 60 percent parallel component for (a) two processing cores and (b) four processing cores.

Q6) Distinguish between parallelism and concurrency.

Q7) Distinguish between data and task parallelism.

3)
**getpid() gets the current process's PID as seen by the OS
**pid is the return value of the fork() sys call, which is different for the parent and child

A: 0
B: 2603
C: 2600
D: 2603

4) Shared memory

5) S = 0.2
gain = 1/(S + (1-S)/N)
   a. g = 1/(0.4 + (0.6)/2) = 1.4286
   b. g = 1/(0.4 + (0.6/4)) = 1.8182

6) Parallel - more than 1 task at a time
Concurrent - more than 1 task making progress at a time

7) Data parallelism - on each core, perform the same operation on subsets of data
Task parallelism - on each core, perform different tasks on the same data
   - threading across cores

Q8) For the program below:

```
int main() {
      Pid_t pid1, pid2;
      Pid1 = fork();
      pid2 = fork();
      if (pid1 == 0) { /* child process */
      pthread_create(. . .);
      }
      if (pid2 == 0) { /* child process */
      pthread_create(. . .);
      }
}
```

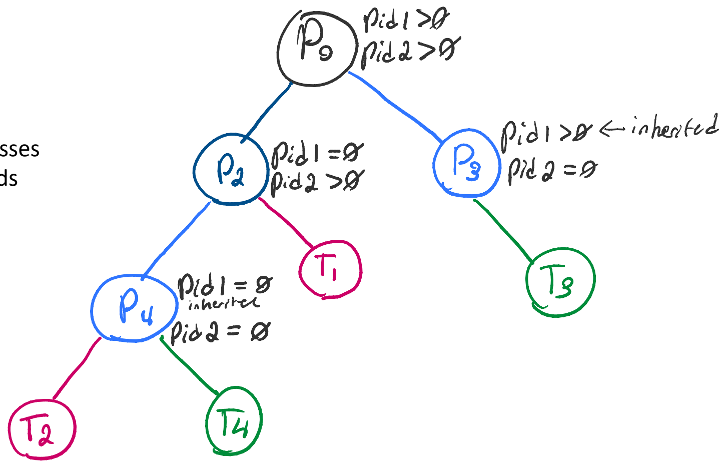
a. How many unique processes are created?
b. How many unique threads are created?
c. Draw the process and thread tree for this code.

Q9) The program shown in Figure 4.16 (page 194 of the textbook) uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

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

int value = 0;
void *runner(void *param); /* the thread */

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

 pid = fork();

 if (pid == 0) { /* child process */
   pthread_attr_init(&attr);
   pthread create(&tid,&attr,runner,NULL);
   pthread_join(tid,NULL);
   printf("CHILD: value = %d",value); /* LINE C */
 }
 else if (pid gt; 0) { /* parent process */
   wait(NULL);
   printf("PARENT: value = %d",value); /* LINE P */
 }
```

8)

a. 4 processes
b. 4 threads



9) C: 5, P: 0

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

Q10) Consider a multicore system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be greater than the number of processing cores in the system. Discuss the performance implications of the following scenarios:

a) The number of kernel threads allocated to the program is less than the number of processing cores.
b) The number of kernel threads allocated to the program is equal to the number of processors.
c) The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user level threads

Q 11) A system with two dual-core processors has four processors available for scheduling. A CPU-intensive application is running on this system. All input is performed at program start-up when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file. Between start-up and termination, the program is entirely CPU-bound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread).

1. How many threads will you create to perform the input and output? Explain.
2. How many threads will you create for the CPU-intensive portion of the application? Explain.

10

a) suboptimal, we want to maximize the use of resources so we should allocate more kernel threads to the program in order to increase throughput
b) optimal, assigning one kernel thread to each processing core maximizes the use of resources
c) has consequences: context switching needs to be performed more often, comes with a lot of overhead

11

a) currently the application is single threaded, and it should remain that way for the input and output so to avoid multiple threads modifying shared resources.
   Since only one file is being read from and written to, we do not need to multithread this task and use synchronization tools to ensure that the shared resources do not get modified incorrectly.
   a. it is simpler and easier if we use one thread for input and one for output
   b. reading/writing is a blocking operation!!!!
b) Since the system has four processors available, we should aim to have 4 threads (user/kernel) in order to maximize resource usage.