CS376 – Operating Systems
Spring 2020
Problem Set #2

Dalton Rothenberger

INSTRUCTIONS:

Answer the following questions as completely as possible. All work must be in one document and submitted to Canvas by the due date. Any drawings may be done by hand but need to be incorporated into the document you submit.

1. *(15 points)* For the following program, determine how many processes are created during its execution. Be sure to include the parent process. Draw a process tree labeling the first fork command process "A" the second fork command process B, and the third, C.

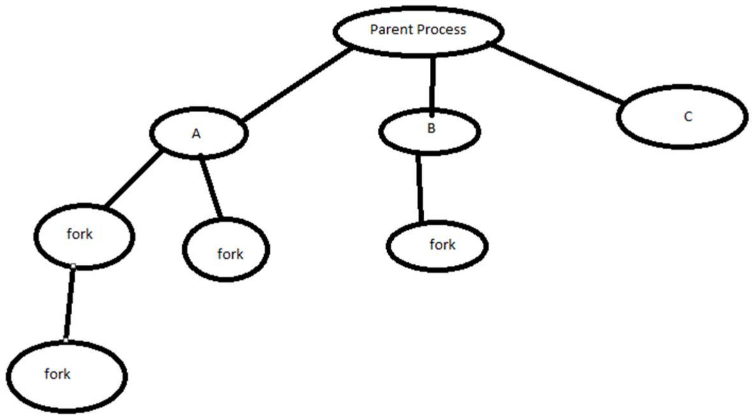
```
#include <stdio.h>
#include <unistd.h>

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

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

   /* and fork another */
   fork();

   return 0;
}
```



8 Processes (Including parent process)

2. *(20 points)* For the following program, identify the values of the pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.) Draw the process tree for this program.

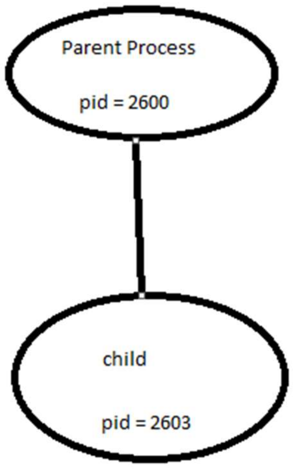
```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid, pid1;

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

   if (pid < 0) { /* error 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;
}
```



A = 0

B = 2603

C = 2603

D = 2600

3. *(15 points)*Using the program shown below, explain what the output will be at Line A.

```
#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;
  }
}
```

5 will be the output. This is because value is initialized to 5. The child process adds 15 to its version of value but this does not impact the value that the parent process has which is the one that is outputting

4. What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level. Be specific and thorough.
    a. *(10 points)* Synchronous and asynchronous communication

For synchronous when sending, the sending process is blocked until the message reaches where it is intended to go. When sending with asynchronous communication, the process sends the message and then goes back to what it was doing. When blocking with synchronous, the receiver is blocked until a message is available whereas with asynchronous, the receiver retrieves a message or a null. Asynchronous is faster but it is less reliable than synchronous. Usually, both types of communication are available in the system.

    b. *(10 points)* Automatic and explicit buffering

With automatic buffering the queue can be potentially infinite so the sender never needs to block for space and any number of messages can wait in the queue. However, with automatic buffering memory is wasted because of the memory reserved for the "infinite" queue. Explicit buffering has a queue of finite length, but this almost means a limit on the number of messages. If the queue is full then the sender needs to block until space becomes available. Explicit buffering does not waste as much memory because of the fixed length.

    c. *(10 points)* Send by copy and send by reference

Send by copy and send by reference act similar to pass by reference and pass by value from Java. Send by copy is when the receiver is not allowed to alter the state of the parameter whereas send by reference does permit it. Send by copy is better for generalization but send by reference is better for larger data structures being sent but has the risk of the structure being altered from various locations.

    d. *(10 points)* Fixed-sized and variable-sized messages

With fixed-sized messages, the system-level implementation is straight forward, but the restriction makes programming more difficult. Variable-sized messages require more complex system level implementation, but programming becomes easier as a result. These also relate back to buffering because with a fixed-size and explicit buffer you know exactly how many messages the queue can hold but with variable-sized messages you have know idea how many can be held.

5. *(10 points)* In the program below, give the output on lines X and Y

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

#define SIZE 5

int nums[SIZE] = {0,1,2,3,4};

int main()
{
int i;
pid_t pid;

  pid = fork();

  if (pid == 0) {
   for (i = 0; i lt; SIZE; i++) {
    nums[i] *= -i;
    printf("CHILD: %d ",nums[i]); /* LINE X */
   }
 }
  else if (pid gt; 0) {
   wait(NULL);
   for (i = 0; i lt; SIZE; i++)
    printf("PARENT: %d ",nums[i]); /* LINE Y */
  }
```

Line X will output: CHILD: 0 CHILD: -1 CHILD: -4 CHILD: -9 CHILD: -16

Line Y will output: PARENT:0 PARENT:1 PARENT: 2 PARENT :3 PARENT: 4

6. Explain the circumstances under which the line of code marked printf("LINE J") in the code below will be reached.

```c
#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>


int main()

{

pid_t pid;


  /* fork a child process */

  pid = fork();


  if (pid lt; 0) { /* error occurred */

    fprintf(stderr, "Fork Failed");

    return 1;

  }

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

    execlp("/bin/ls","ls",NULL);

    printf("LINE J");

  }

  else { /* parent process */

    /* parent will wait for the child to complete */

    wait(NULL);

    printf("Child Complete");

  }


return 0;
```

Line J will be reached in the child process as long as an error did not occur when creating the child process, because the child process will have pid == 0.