CS433 Written Homework 2

**Written questions (60 points)**

All question numbers refer to exercises in the textbook (eBook 10th edition). Make sure you use the right textbook and answer right questions. Students need to finish written questions individually. Type your answers and present your answers and steps clearly.

1. (5 points) 3.11 Including the initial parent process, how many processes are created by the program shown in Figure 3.32? Explain your answer.

The answer is 15 processe. Each time one parent forks into two childen those two children will fork again(up to the fourth fork) which is the same as 2^4-1 which is equal to 15(minus one because there is only one initial parent)

  #include <stdio.h>

  #include <unistd.h>

  {

  int main()

  {

    int i;

  {

    for (i = 0; i < 4; i++)

     fork();

  {

    return 0;

  }

[**Figure E3.32**](https://jigsaw.vitalsource.com/books/9781119320913/epub/OPS/c03.xhtml#c03-fig-5055-a) How many processes are created?

1. (10 points) 3.13 Using the program in Figure E3.34, 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.)

A = 0

B = 2603

C = 2603

D = 2600

#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;

}

[**Figure E3.34**](https://jigsaw.vitalsource.com/books/9781119320913/epub/OPS/c03.xhtml#c03-fig-5059-a) What are the pid values?

1. (10 points) 3.16 Using the program shown in Figure E3.35, explain what the output will be at lines X and Y.

X = 0, -1, -4, -9, -16

Y = 0, 1, 2, 3, 4

#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 \*/

}

}

return 0;

}

[**Figure E3.35**](https://jigsaw.vitalsource.com/books/9781119320913/epub/OPS/c03.xhtml#c03-fig-5063-a) What output will be at Line X and Line Y?

4. (10 points) **3.17** What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.

a. Synchronous and asynchronous communication

b. Automatic and explicit buffering

c. Send by copy and send by reference

d. Fixed-sized and variable-sized messages

5. (5 points) **4.10** Which of the following components of program state are shared across threads in a multithreaded process?

1. Register values
2. Heap memory
3. Global variables
4. Stack memory

Heap memory and Global variable are the only two components here shared across multithreaded processes

6. (10 points) **4.17** Consider the following code segment:

  pid\_t pid;

  pid = fork();

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

    fork();

    thread\_create( . . .);

  }

  fork();

1. How many unique processes are created?

6 processes are created

1. How many unique threads are created?

2 threads are created

7. (5 points) **4.19** The program shown in [Figure E4.23](https://jigsaw.vitalsource.com/books/9781119320913/epub/OPS/c04.xhtml#c04-fig-5094) uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

C = 5

P = 0

  #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 \*/

   }

  }

  void \*runner(void \*param) {

     value = 5;

     pthread\_exit(0);

  }

[**Figure E4.23**](https://jigsaw.vitalsource.com/books/9781119320913/epub/OPS/c04.xhtml#c04-fig-5094-a) C program for [Exercise 4.19](https://jigsaw.vitalsource.com/books/9781119320913/epub/OPS/c04.xhtml#c04-ex-5093).

8. (5 points) **4.20** 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.

1. The number of kernel threads allocated to the program is less than the number of processing cores.

There will be idle processors because there are no kernel threads to map to them

1. The number of kernel threads allocated to the program is equal to the number of processing cores.

All processors will be functioning and in use unless a kernel thread is blocked

1. The number of kernel threads allocated to the program is greater than the number of processing cores but less than the number of user-level threads.

All processors will be functioning and in use, and if a kernel thread is blocked it can be replaced with another