Question

Assume several threads are created using the many-to-one threading model. Discuss whether they can execute concurrently, in parallel or both.

Answer

Only concurrently, in parallel requires distinct kernel threads (many-to-one means threads stay in user space)

Question

What is the pthread_join function used for? Write a small sample code to support your discussion.

Answer

To stop a thread execution until another thread has completed, to join a thread.
About the code: the idea is that students can write something like this

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

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
  pthread_t tid; /* the thread identifier */
  pthread_attr_t attr; /* set of thread attributes */

  if (argc != 2) {
    fprintf(stderr,"usage: a.out <integer value>\n");
    return -1;
  }
  if (atoi(argv[1]) < 0) {
    fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
    return -1;
  }
```

```
    /* get the default attributes */
    pthread_attr_init(&attr);
    /* create the thread */
    pthread_create(&tid,&attr,runner,argv[1]);
    /* wait for the thread to exit */
    pthread_join(tid,NULL);

    printf("sum = %d\n",sum);
  }

  /* The thread will begin control in this function */
  void *runner(void *param)
  {
    int i, upper = atoi(param);
    sum = 0;

    for (i = 1; i <= upper; i++)
      sum += i;

    pthread_exit(0);
  }
```

What the students should show through their code is that they are aware that each thread has an id, that you need to keep track of it to call join, that you need to pass parameters when you create a thread about its attributes, as well as the function to be run and that you need to join, especially if you are then trying to operate on a shared variable.

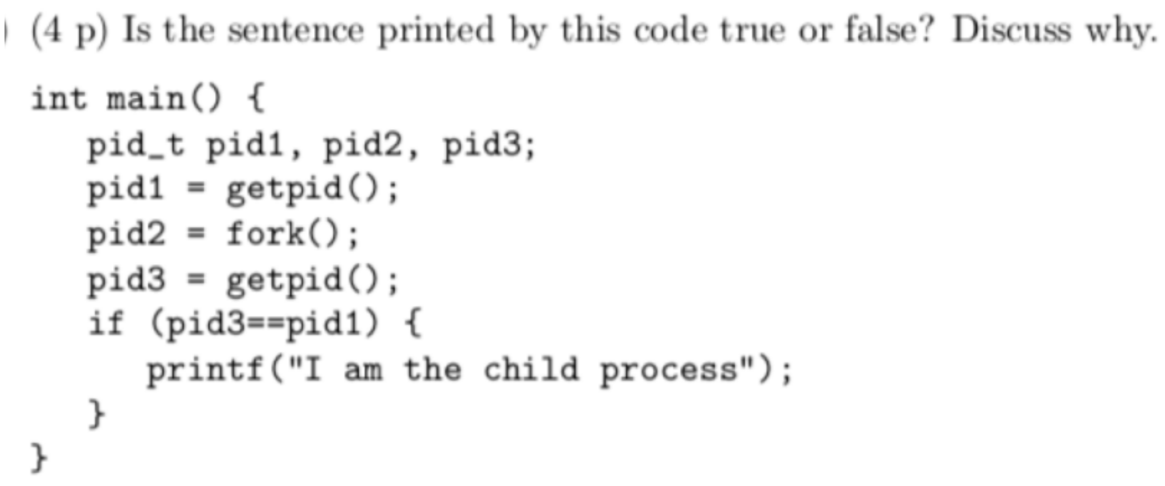
Question

Explain why concurrent and parallel execution can be achieved by a program by means of both multiple processes and multiple threads.

Answer

Here the point is that if you want to use multiple cores to run a program, you can design the program to have multiple processes that communicate with IPC. Might be not as efficient as threads, but it works (and is modular, good from fault tolerance perspective). Multithreaded processes are not what enables the leveraging of multi-core architectures.

Question

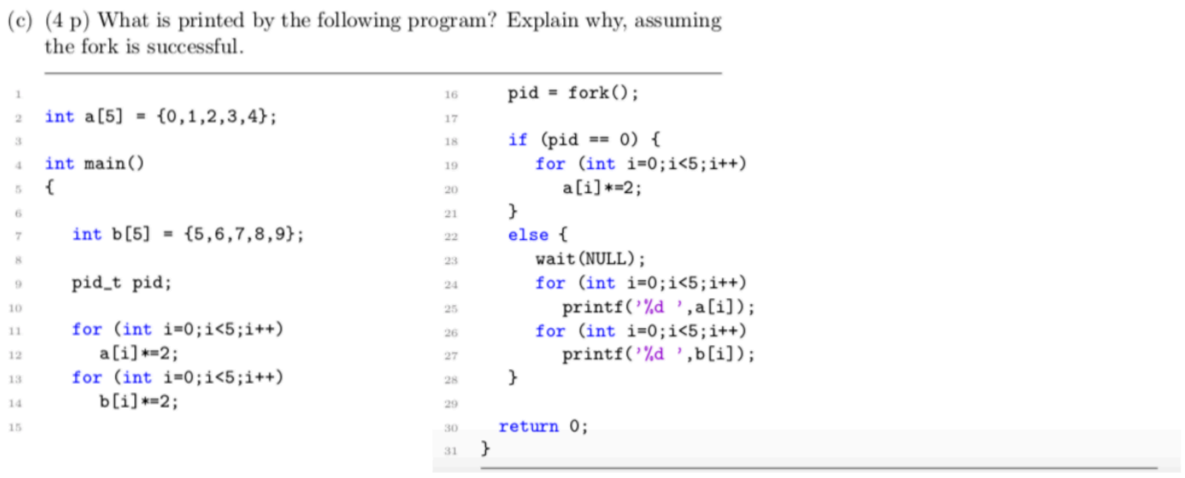

(4 p) Is the sentence printed by this code true or false? Discuss why.

```
int main() {
   pid_t pid1, pid2, pid3;
   pid1 = getpid();
   pid2 = fork();
   pid3 = getpid();
   if (pid3==pid1) {
      printf("I am the child process");
   }
}
```

Answer

False, the child will have a different pid than the parent, and since pid1 is set by the parent and pid3 by the child the sentence is false;

Question


(c) (4 p) What is printed by the following program? Explain why, assuming the fork is successful.

```

int a[5] = {0,1,2,3,4};

int main()
{

   int b[5] = {5,6,7,8,9};

   pid_t pid;

   for (int i=0;i<5;i++)
      a[i]*=2;
   for (int i=0;i<5;i++)
      b[i]*=2;

   pid = fork();

   if (pid == 0) {
      for (int i=0;i<5;i++)
         a[i]*=2;
   }
   else {
      wait(NULL);
      for (int i=0;i<5;i++)
         printf('%d ',a[i]);
      for (int i=0;i<5;i++)
         printf('%d ',b[i]);
   }

  return 0;
}
```

Answer

Whatever was in the arrays before the fork, the child modifies its own copy, so even if the parent waits the change is not reflected in its data.