

# Pre-Lecture 12

**Due** Oct 2 at 9am**Points** 8**Questions** 6**Available** until Oct 2 at 9am**Time Limit** None**Allowed Attempts** 2

## Instructions

Take this quiz *after you have watched the required videos and/or read the associated sections of the textbook*. See [Lecture 12: Processes II](#).

You may attempt this quiz twice. Incorrect responses are marked after each attempt. Correct answers are revealed at the start of class for this lecture.

Carefully note the deadline for responses. Submissions are not accepted after the deadline, and there is no grace period.

This quiz was locked Oct 2 at 9am.

## Attempt History

	Attempt	Time	Score
<b>LATEST</b>	<a href="#">Attempt 1</a>	127 minutes	8 out of 8

Score for this attempt: **8** out of 8

Submitted Oct 1 at 4:26pm

This attempt took 127 minutes.

### Question 1

**1 / 1 pts**

What is printed by the following C program?

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

int main() {
    pid_t pid = fork();
    if(pid == 0)
        exit(42);
    else {
        int status = 0;
        waitpid(pid, &status, 0);
        printf("%d", WEXITSTATUS(status));
    }
}
```

**Correct!**

42

**Correct Answers**

42

**Question 2****1 / 1 pts**

When a parent process that has forked a long-running child process exits, it causes the child process to end prematurely.

☐ True**Correct!**☒ False**Question 3****1 / 1 pts**

When can a process ID be reused? (Select all that apply.)

☐ When the process with that ID has exited with status 0.**Correct!**

When the process with that ID has exited and been reaped by its parent using *waitpid*.



When the process with that ID has been running for more than ten minutes.

☐ When all process IDs 1 to 32000 have been used.**Question 4****1 / 1 pts**

What is the ID of the *init* process?

**Correct!**

1

**Correct Answers**

1

**Question 5****3 / 3 pts**

Consider the following C program:

```
#include "csapp.h"

/* Wait() is the same as Waitpid() with the
   pid and options parameters set to defaults.
   That is, it blocks until any child terminates. */

int main() {
    if(Fork() == 0) {
        if(Fork() == 0)
            printf("a");
        else {
            pid_t pid;
            int status;
            if((pid = Wait(&status)) > 0)
                printf("b");
        }
    }
    else {
        printf("c");
        exit(0);
    }
    printf("d");
    return 0;
}
```

Which of these outputs is possible? (Select all that apply.)

☐ abddc**Correct!**☒ acdbd**Correct!**☒ adbdc☐ bdadc**Correct!**☒ cadbd

**Question 6****1 / 1 pts**

Consider the following C program *parent.c* (compiled to executable *parent*):

```
#include <unistd.h>

extern char** environ;

int main(int argc, char** argv) {
    argv[0] = "child";
    execve("child", argv, environ);
    return 0;
}
```

Also, consider the following C program *child.c* (compiled to executable *child*):

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

int main(int argc, char** argv) {
    int i;
    for(i = 0; i < argc; i++)
        printf("%s ", argv[i]);
    printf("%d", getpid());
    return 0;
}
```

Suppose that we run the parent program with this command:

```
./parent bird cat dog
```

Also, suppose that we use *ps* while the program is running to determine that the PID of *parent* in execution is 24712.

Exactly what is printed?

**Correct!**

```
child bird cat dog 24712
```

**Correct Answers**

```
child bird cat dog 24712
```

**Quiz Score: 8 out of 8**