# Preview Test: Homework-Ch3

## Test Information

Description

Instructions

Multiple Attempts Not allowed. This test can only be taken once.

Force Completion This test can be saved and resumed later.

### QUESTION 1

1 points Saved

True or False (1 point): On function return, the OS automatically deallocates the stack space assigned to the function.

- ○ True
- ◉ False

### QUESTION 2

1 points Saved

True or False (1 point): On Linux, the parent process by default waits for its child process to finish.

- ○ True
- ◉ False

### QUESTION 3

1 points Saved

True or False (1 point): In the *shared memory* IPC model, "reads" to shared memory are typically blocking.

- ○ True
- ◉ False

### QUESTION 4

1 points Saved

True or False (1 point): Any two processes running on the same OS can use the (anonymous) pipe IPC mechanism to communicate with each other.

- ○ True
- ◉ False

### QUESTION 5

1 points Saved

True or False (1 point): The *shared memory* IPC model will typically allow faster communication than *message passing*.

- ◉ True
- ○ False

### QUESTION 6

1 points Saved

Fill-in the blank (1 point). The answer can only be a single upper-case letter from 'A' - 'C'.

The following IPC mechanism allows sent messages to be received out-of-order -- C

Options are: 'A' - Anonymous Pipe ; 'B' - Fifo ; 'C' Message queue

Question Completion Status:

## QUESTION 7

2 points Saved

Fill-in the blanks (1 point each): Each answer should only be a single-letter option from 'A' - 'D' (upper-case and without quotes).

The process address space is divided into 4 regions: 'A' - Stack, 'B' - Heap, 'C' - Data, 'D'- Text

Answer the following questions regarding the process-address space using an option from 'A' to 'D'.

Questions:

1. A dynamically allocated variable is always assigned space from the B region of the process address space.

2. The program counter (PC) points into the D region of the process address space.

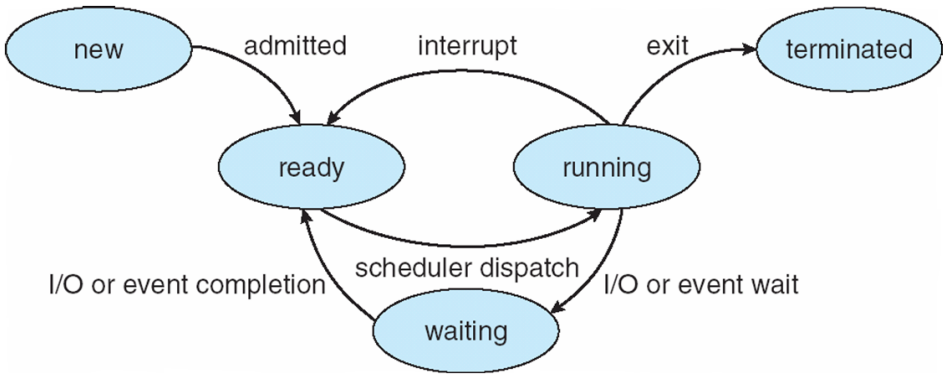
## QUESTION 8

2 points Saved

Fill-in the blanks (1 point each): Each answer should only be a single-letter option from 'A' - 'G' (upper-case and without quotes).

Using the figure below, indicate the transition between process states that will happen for the specified process on the given events. Options are:

'A' = Admitted, 'B' - Interrupt, 'C' - Exit, 'D' - I/O or event completion, 'E' - I/O or event wait, 'G' - Scheduler dispatch




Questions:

1. For the currently running process when it issues the *printf* command to write a message to the screen -- D

2. For the process that is scheduled by the OS to run on the CPU in a time-shared OS -- G

## QUESTION 9

2 points Saved

Answer the questions with a 'T' or 'F', upper-case and without the quotes (1 point each):
Consider the following program:

```
int main(){
   pid_t pid;

   printf("Process id is: %d\n", pid);
   pid = fork();     /* fork another process */

   if (pid == 0) { /* child process */
      execlp("/bin/ls", "ls", NULL);
      printf("Process exiting: %d\n", getpid());
   }
 if (pid > 0) { /* parent process */
 wait(NULL);
      printf ("Process exiting: %d\n", getpid());
   }

   exit(0);
}
```

Assume that,
(a) the "fork" and "exec" calls are successful,
(b) after the fork, the parent process runs before the child process, and

Question Completion Status:

Questions:

1. This program will print the line "Process id is: 11" -- F

2. The line "Process exiting: 11" is printed before the line "Process exiting: 10" -- F

**QUESTION 10**

3 points Saved

Fill-in the blanks (0.5 point each): Each answer should only be a single-letter option from 'A' - 'F' (upper-case and without quotes). The options can be repeated.

For the program given below, indicate what code will you insert at the marked spots to use the pipe IPC mechanism to synchronize the parent and child processes
From Child process
From Parent process

Options are: 'A' - pipe(fds) ; 'B' - read(fds[0], ...) ; 'C' - read(fds[1], ...) ; 'D' - write(fds[0], ...) ; 'E' - write(fds[1], ...) ; 'F' - Nothing to write here

Program:

```
int main(){

   char *s, buf[1024];
   int fds[2];
   char *s = "Pipe program for process synchronization\n";

A

   if (fork() == 0) {

F
      printf("From Child process\n");
E

   } else {

B
      printf("From Parent process\n");
F
   }

F
}
```

*Click Save and Submit to save and submit. Click Save All Answers to save all answers.*

Save All Answers Save and Submit