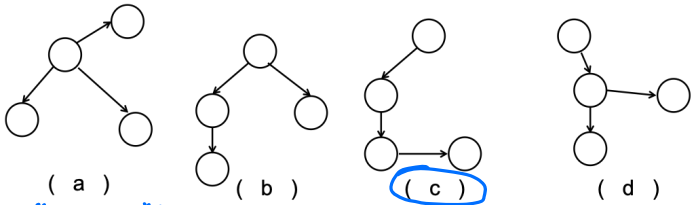
CSE4001 (Spring 2022) Exam 1

Student name:

---

1. What graph illustrates the process-creation pattern described by the following program?

```
int i, n = 4;
pid_t pid;
for ( i = 1; i < n; ++i )
   if ( pid = fork() )
      break;
}
```

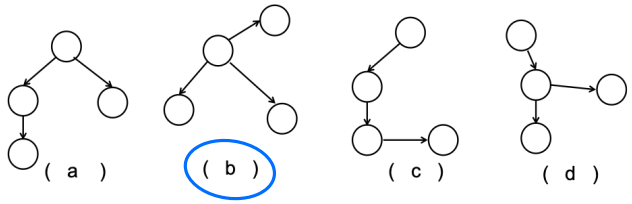


Create child. If "parent" break from loop. If child, iterate.

2. What graph illustrates the process-creation pattern described by the following program?

```
int i, n = 4;
pid_t pid;
for ( i = 1; i < n; ++i )
   if ( ( pid = fork() ) <= 0 )
      break;
}
```

If "child", break. If "parent", iterate.

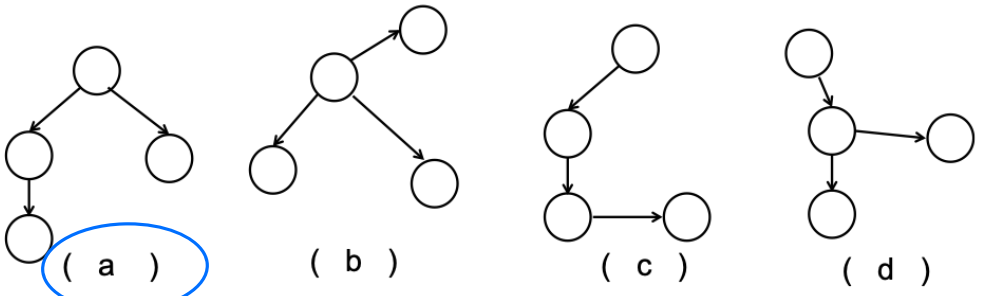


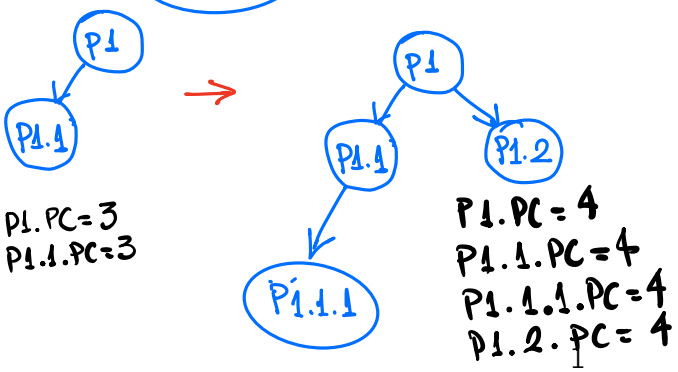
3. What graph illustrates the process-creation pattern described by the following program?

```
int main() {
   int x = 0, y = 1;
   fork(); printf( "x = %d\n", x );
   y = x + 1;
   fork(); printf( "y = %d\n", y );
}
```

==

```
1  int main() {
2     fork();
3     fork();
4  }
```





4. Consider the user-level function `int add_two_numbers(int a, int b)` to be implemented as a system call in OS/161. How should the corresponding kernel-level implementation be called in the OS/161's system-call handler?

   (a) `int retval = add_two_numbers(int a, int b)`

   (b) `int retval = sys_add_two_numbers(int a, int b)`

   (c) `int retval = add_two_numbers(tf->tf_a0, tf->tf_a1)`

   (d) `err = add_two_numbers(tf->tf_a0, tf->tf_a1, &retval)`

   (e) `err = sys_add_two_numbers(tf->tf_a0, tf->tf_a1, &retval)`

   - must have sys_ prefix
   - return value is "err"
   - sum "returns" by reference.

5. Which function in the process API has code that kills the zombi child process?

   (a) `exec()`

   (b) `wait()` ← read book for more info.

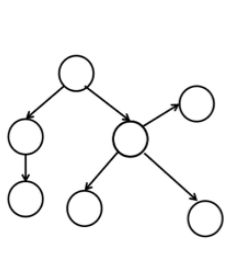
   (c) `fork()`

   (d) `getppid()`

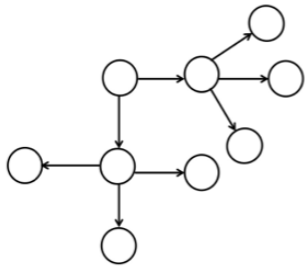
   (e) `exit()`

6. What graph illustrates the process-creation pattern described by the following program?

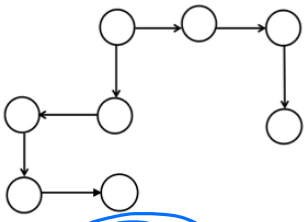
```
int main() {
  int i, n = 3;
  pid_t pid;
  fork();
  for ( i = 1; i < n; ++i )
     if ( pid = fork() )
        break;
}
```



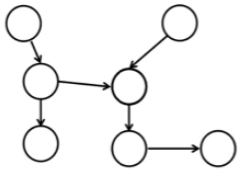
(a)



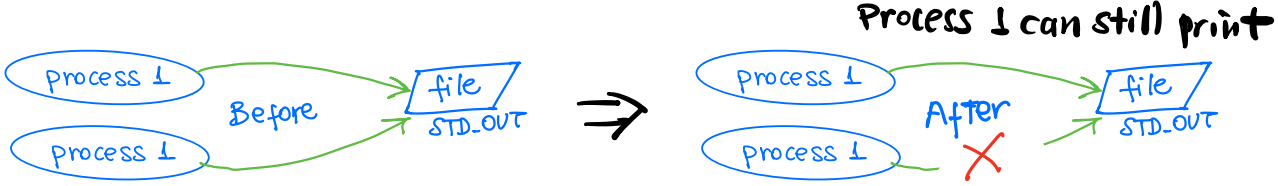
(b)



(c)



(d)



7. Consider a program that first creates a child process, and then the child closes standard output (`STDOUT_FILENO`). Can the parent print something using `printf` after the child process closes `STDOUT_FILENO`?

   (a) Both the child and parent can print to the console.

   (b) Only the parent can print to the console.

   (c) Only the child can print to the console.

   (d) Neither parent nor child can print to the console.

   (e) The child will crash if it attempts to print to the console linked to `STDOUT_FILENO`.

8. Consider the following program. What is the output in the lines indicated by A, B, C, and D? (Assume that the actual PIDs of parent and child are 10622 and 10623, respectively).

```
int main(){
   pid_t pid, pid1;
   pid = fork();
   if ( pid ) {
      pid1 = getpid();
      printf( "pid = %d\n", pid );   /*   A   */
      printf( "pid = %d\n", pid1 );  /*   B   */
   } else {
      pid1 = getpid();
      printf( "pid = %d\n", pid );   /*   C   */
      printf( "pid = %d\n", pid1 );  /*   D   */
      wait( NULL );
   }
   return 0;
}
```

This is run by parent. pid is non-zero for parent

This part is run by child because variable pid is zero.

   (a) `pid = 0, pid = 10623, pid = 10622, pid = 10623.`

   (b) `pid = 10623, pid = 10622, pid = 0, pid = 10623.`

   (c) `pid = 10623, pid = 10622, pid = 0, pid = 10622.`

   (d) `pid = 0, pid = 10622, pid = 10622, pid = 10623.`

   (e) `pid = 10622, pid = 0, pid = 0, pid = 10623.`

9. In OS/161, which function sets the trapframe?

Trapframe stores the content of registers when context switch occurs due to exception or interrupt. First program to run is the assembly function that set up the trapframe and calls the trap handler.

(a) `mips_trap()`

(b) `syscall()`

(c) `exception-mips1.S`

(d) `trapframe`

(e) `main()`

10. Consider the following program. Assuming the the `exec` call completed successfully, when is the value of `outvar` printed by the child process when it executes the line `printf("Output = %d", outvar)`?

```
int main(){
   outvar = 10;
   pid_t  pid = fork();
   if (pid == 0) {
      execlp("/bin/ls", "ls", "-l", NULL);
      printf("Output = %d", outvar);
   } else {
      wait (NULL);
      exit(0);
   }
}
```

Never, because this code no longer exists in the process address space

(a) 0

(b) 10

(c) Right after the completion of `ls`

(d) Never

(e) Program goes into an infinite loop because that `printf` statement is no longer valid

11. What happens when a process calls `exec()`?

(a) A new process is created. This new process loads the executable code of another program and runs it.

(b) The process that called `exec()` has its code replaced by the code of another program. The new code will execute from its first instruction when the scheduler selects the process to run.

(c) The code of the process that called `exec()` runs the code another program just like a function call. Once the new code reaches its end, the flow of execution returns to the next instruction following the `exec()` call in the calling process.

(d) The code of the process that called `exec()` is replaced by the code of another program. Once this program ends, the control returns to the next instruction following the `exec()`.

(e) None of the above.

12. Consider the following program. The program has two calls for `wait`, one of which is from the child process. What happens when this program executes?

```
int main(){
   outvar = 10;
   pid_t  pid = fork();
   if (pid == 0) {
      printf("Output = %d", outvar);
      wait (NULL);
   } else {
      wait (NULL);
      printf("Done.");
      exit(0);
   }
}
```

(a) The child process prints the `Output = 10` to the console. Because the child process has no children, the call to wait will block the process which will wait forever for the child that does not exist. As a result, the parent process will never print its message, i.e., `Done`.

(b) The child process prints the `Output = 10` to the console first and terminates without any major issues. Then, the parent process prints `Done`.

(c) The child crashes. The parent waits forever for the child.

(d) The child crashes. The parent prints its output, i.e., `Done`.

(e) If the parent executes first then the parent crashes because the child process hasn't executed yet.

13. Which program contains the general interrupt handler in OS/161?

(a) `mips_trap()`

(b) `syscall()`

(c) `exception-mips1.S`

(d) `trapframe`

(e) `main()`

14. In OS/161, which program implements the system-call handler?

(a) `mips_trap()`

(b) `syscall()`

(c) `exception-mips1.S`

(d) `trapframe`

(e) `main()`

15. Consider the following program. Which of the following alternatives is **wrong**?

```
int main(){
   outvar = 10;
   pid_t  pid = fork();
   if (pid == 0) {
      execlp("/bin/ls", "ls", "-l", NULL);
   } else {
      exit(0);
   }
}
```

Without wait(), we can't tell the order of execution. Scheduling is not first-come first-served.

(a) The parent process may terminate before the child runs.

(b) Both parent and child processes can execute and terminate in any order.

(c) When `fork()` creates the child process, the state of the child process is set to `READY` while the parent process is still `BLOCKED` waiting for the `fork` to finish. In this case, the child process will execute first and then the parent process will execute afterwards.

(d) If the parent process terminate first, the child process can still run.

(e) The address spaces of the processes are not shared.

16. Consider the following code:

```
int main(){
   int x = 0, y = 1;
   fork();
   y = x + 1;
   printf( "x = %d\n", x );
}
```

Only one of the following statements is correct. Select the correct statement.

(a) There are two processes and they do not share variables at all.

(b) There is just one process.

(c) There are three cooperating processes and they do not share the variable `x`.

(d) There are two processes and they both write on the shared variable `x`. ?

(e) There is just one process but variable x remains unmodified after the fork call.

17. In the implementation of `sys__exit()` in OS/161, what is the main function that is called to terminate the thread associated to the process being exited?

    (a) `void thread_exit(void)`

    (b) `void thread_destroy(struct thread *thread)`

    (c) `void exorcise(void)`

    (d) `void thread_panic(void)`

    (e) `void thread_shutdown(void)`

18. What are all the possible outputs (as printed by `printf`) of the following program? Assume that the parent process has PID 1223, and that the child process has PID 1224. Assume that the `fork()` will be successful.

```
int main(int argc, char *argv[]){
   printf("hello world (pid:%d)\n", (int) getpid());
   int rc = fork();
 if (rc == 0) {
      printf("I am child (pid:%d) ", (int) getpid());
   } else {
      printf("I am parent of %d (pid:%d) ", rc, (int) getpid());
   }
   return 0;
}
```

    (a) `I am child (pid:1224) I am parent of 1224 (pid:1223)` or `I am parent of 1224 (pid:1223) I am child (pid:1224)`

    (b) `I am parent of 1224 (pid:1223)`

    (c) `I am child (pid:1224)`

    (d) `I am parent of 1224 (pid:1223)` followed by `I am child (pid:1224)`

    (e) `I am child (pid:1224)` followed by `I am parent of 1224 (pid:1223)`

19. Consider a program that opens a file (i.e., with the open() system call) and then calls `fork()` to create a new process. (a) Can both the child and parent write to the file? (b) If the parent process closes the file (e.g., using `fclose()`), can the child process still write on the file?

    (a) Both the child and parent can write to the file.

    (b) Only the parent can write on the file.

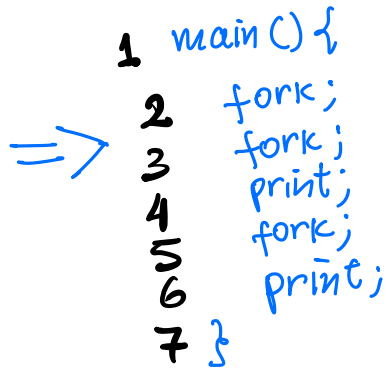
    (c) Only the child can write on the file.

    (d) Neither parent nor child can write to the file.

    (e) The parent writes to the file first and then the child can write to the file.

20. What is the maximum number of "hello"s printed by the following program?

```
#include <stdio.h>
#include <unistd.h>
main()
{
  int i;
  fork();
  for (i=0; i<2; i++){
    fork();
    printf("hello\n");
  }
}
```



(a) 3

(b) 4

(c) 7

(d) 8

(e) 2

21. What is the expected output of the following program? Assume that `fork`is successful.

```
int main (int argc, char *argv[]) {
    int pid = fork();
    if (pid < 0) {
       fprintf(stderr, "error: fork failed.\n");
       exit(1);
    } else if (pid==0) {
       close(STDOUT_FILENO);
       printf("child ");
    } else {
       printf("parent ");
    }
    return 0;
}
```

(a) child parent

(b) parent child

(c) child

(d) parent

(e) No output