CSCI-202 Homework 2

Petr Peganov pvp6348

February 2024

Problem 1

a) $thread\_create(func)$ starts a new thread in the existing process which shares the processes memory. This thread begins execution at func.
fork() creates a new process, which is a copy of the parent process, but it has a separate space in memory.
b)Process: Stores information about the environment like global variables and memory space.
Thread: It shares resources with its process and stores only thread-specific information; also, it requires a stack.

Problem 2

```
// find_insert_pos
node_t *current = head;
while (current->next != NULL && current->next->id < node->id) {
    current = current->next;
}
return current;

//insert
if (pos == NULL) {
    node->next = head;
    head = node;
} else {
    node->next = pos->next;
    pos->next = node;
}
```

Problem 3

3.1:
echo echo hello $world outputs echo hello.

echo 'echo hello $world' outputs echo hello $world.
echo "echo hello $world" outputs echo hello plus the value of $world.
echo `echo hello $world` executes echo hello $world and then echo the result, printing hello.
echo (echo hello $world) will give a syntax error.

3.2:
echo 'hello world' — cat prints hello world.
echo 'hello world' > cat creates a file named cat with hello world.
echo 'hello world' 2 > cat creates a file called cat but it will be empty.

3.3:
echo a && echo b prints a then b but only if the first command is successful.
echo a ; echo b outputs outputs a then b without regard to the success of the first command.
echo a & echo b executes a in the background and b in the foreground.

Shell Pipeline:
First 100 names:

```
grep "^Name:" members.txt | cut -d':' -f 2 | head -n 100
```

Sort and save:

```
grep "^Name:" members.txt | cut -d':' -f 2 | sort | head -n 100 | tee names.txt
```

Problem 4

There are 3 possible outputs:

```
1.
    foo: 0
    boo: 1
    main: 2
2.
    boo: 0
    foo: 1
    main: 2
3.
    foo: 0
    boo: 0
    main: 2
```

To avoid race conditions, we can use mutex to make sure that only 1 thread at a time modifies i.
After using mutex the possible outputs are:

1.
```
foo: 0
boo: 1
main: 2
```
2.
```
boo: 0
foo: 1
main: 2
```