1)

The kernel does not keep track of user threads, while it does keep track of its own threads. The kernel also does not schedule user threads. A user thread belongs to a process, which is easier to maintain when scheduling.

2)

In first come first serve, short jobs could be hurt if they come after a long job, because then they will have to wait a long time. In round robin, all jobs recieve equal time share of operating time. In multilevel feedback queues, preference is given to shorter jobs, so that they do not have to wait behind long jobs.

3)

At the beginning one processes starts the for loop. Then it hits fork() on i=0, creating 2 processes.

Fork fork

F f f f

F f f f f f f f

p p p p p p p p p p p p p p p p

So in the end there will be 16 ‘return 0’s .

4)

Neither the stack nor the heap is shared between the split processes. Each will have a copy made.

5)

//Jason Gilman

//126006979

#include <string>

#include <stdio.h>

#include <unistd.h>

#include <iostream>

using namespace std;

int main(int argc, char \*argv[])

{

string input = argv[1];

cout << "String input: " << input << endl;

int length = input.length();

pid\_t pid;

for(int i = length-1;i >= 0;i--)

{

pid = fork(); //If you cannot create processes inside the loop, how do you iterate through the string?

if(pid == 0) //returned to the child.

{

cout << input.at(i);

break; //break stops forked process from creating childern.

}

}

}