Introduction to Multithreading
Lab 03
E/15/271

Exercise1

a) Why do you need '- pthread ' flag when you compile?

Add support for multithreading using the POSIX threads library. This option sets flags for both the preprocessor and linker. This option does not affect the thread safety of object code produced by the compiler or that of libraries supplied with it.

If the -pthread flag is not used, the following errors occur in compilation.
/tmp/cckjy03h.o: In function `main':
thread.c:(.text+0x68): undefined reference to `pthread_create'
thread.c:(.text+0x91): undefined reference to `pthread_join'
collect2: error: ld returned 1 exit status

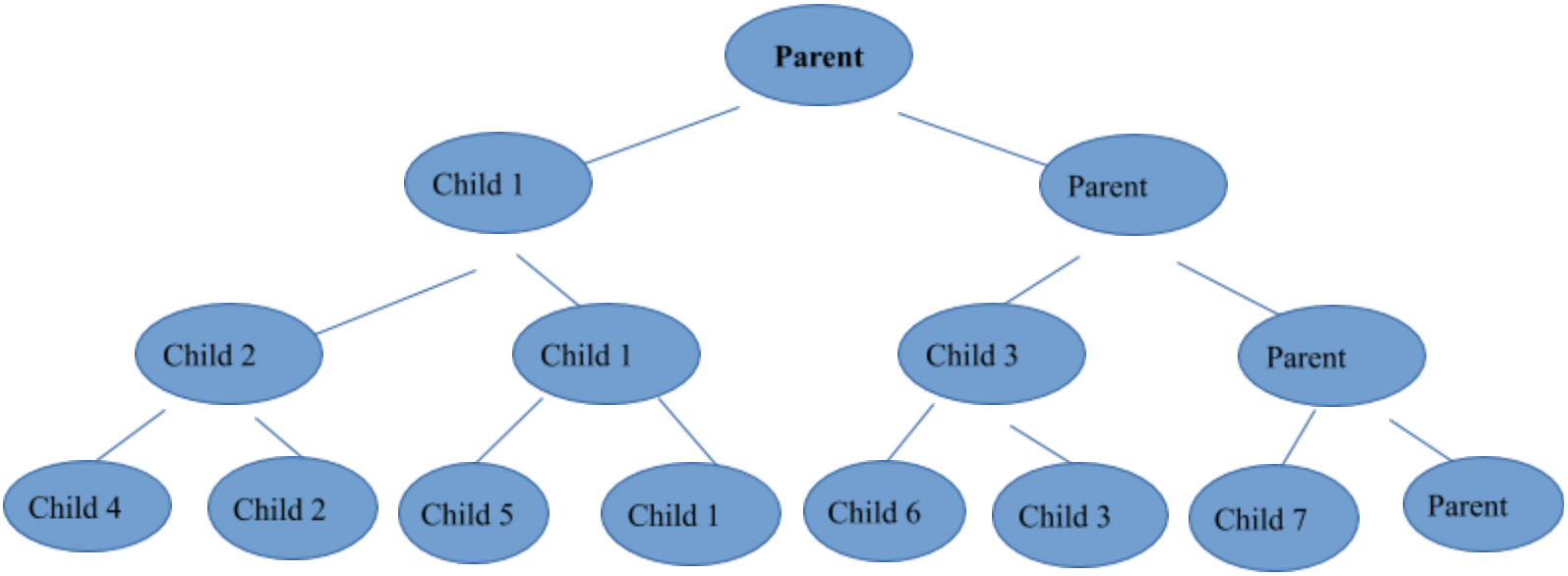
Using the -pthread flag will direct the compiler to compile the pthread library so that the code can use functions from it.

Exercise2

1. Consider the following piece of code from multiprocessing lab session:

```
int i;
for (i = 0; i < 3; i++)
fork();
```

a) how many new processes did it create?



7 processes were created
Formula: 2^n - 1, where n is the number of times that fork() is called. In this case $2^3 – 1 = 8 – 1 = 7$

b) how many threads will it create? Compare this with a) above.

3 new threads created.
With the main thread the total number of threads is 4.

Why is there a difference?

Because each process provides the resources needed to execute a program. A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution. Each process is started with a single thread, often called the primary thread, but can create additional threads from any of its threads.

But a thread is an entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources. In addition, each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled. The thread context includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process.

Exercise 3

1. Compile and run the above program as is.

```
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ gcc -pthread -o ex3 ex3.c -Wall
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ ./ex3
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 2:1 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Thread 3:1 says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 4:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!
Thread 5:1 says hi!
Thread 5:2 says hi!
Thread 5:3 says hi!
Main thread says hi!
```

a) Can you explain the results?

Main thread creates 5 threads sequentially (they begin and complete execution one after the other). The reference to the thread number is passed into the first thread when the value is 1. Every thread increments the value by 1. This is how different threads know their thread numbers. The sequence number within the thread is iterated by the individual threads.

b) What is the objective of the code in line 15?

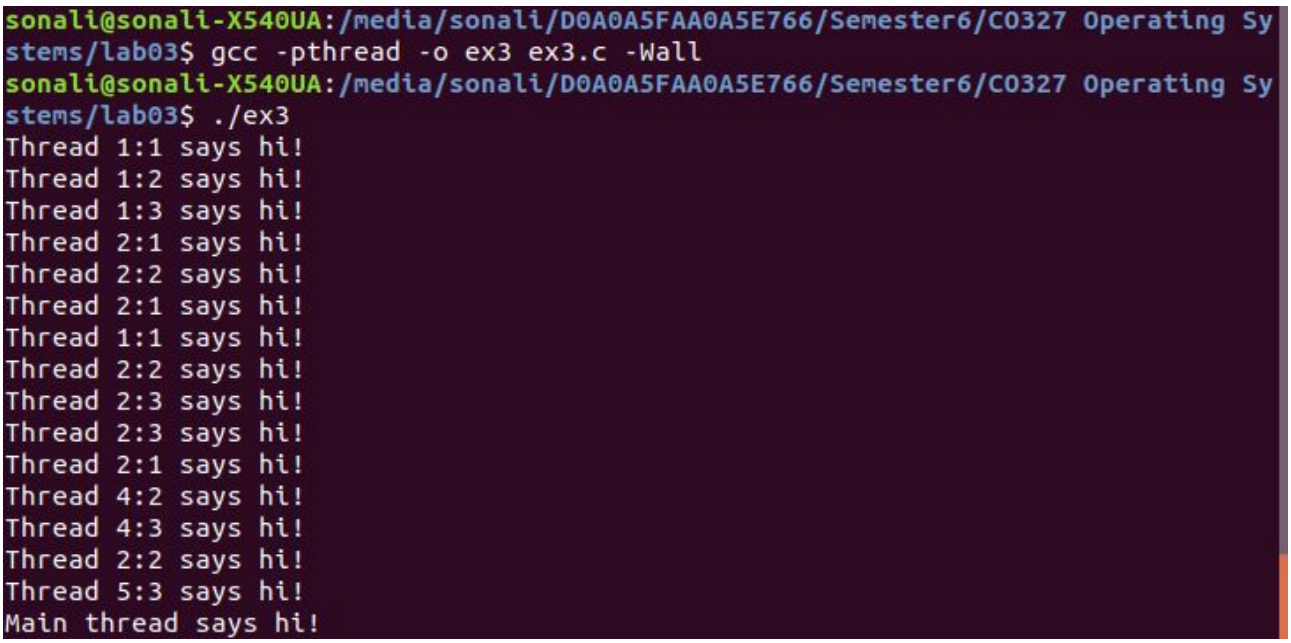
The reference of the memory location with thread number is passed to threads' running function "thread_function()" casted to a void pointer. Because the increment operation cannot be done on the void data type.

c) Deconstruct this statement, and explain how the objective you mentioned is achieved.

The following steps happen in the 15 th line

1. The void* is casted to int*.
2. Next the int* is dereferenced to obtain the value.
3. Then the int vale is incremented.

2. Comment out the code segment for the join call (lines 30 – 34). Can you explain the result?



The threads work asynchronously and threads run in parallel. The number of threads created are still 5.

a) Now, comment out the sleep() statements at lines 12, 35 and 37 (only one at a time) and explain each result.

Commenting out line 12

```
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ gcc -pthread -o ex3 ex3.c -Wall
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ ./ex3
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 2:1 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Thread 3:1 says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 4:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!
Thread 5:1 says hi!
Thread 5:2 says hi!
Thread 5:3 says hi!
Main thread says hi!
```

The program behaves as the original except for there is no delay between all the lines printed for a single thread.

Commenting out line 35

```
Tsonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating S
stems/lab03$ gcc -pthread -o ex3 ex3.c -Wall
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ ./ex3
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:2 says hi!
```

The time between thread creation is removed. Therefore all threads are initialized with thread count 1 before at least one thread prints a single line. Since all threads are of the same thread number. And all 5 threads will print with thread numbers 1 and their internal iterations as 1,2,3.

Commenting out line 37

```
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ gcc -pthread -o ex3 ex3.c -Wall
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ ./ex3
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 1:1 says hi!
Thread 1:3 says hi!
Thread 2:2 says hi!
Thread 2:1 says hi!
Thread 2:3 says hi!
Thread 3:2 says hi!
Thread 3:1 says hi!
Main thread says hi!
```

Since there is no sleep between starting the last thread and the server thread printing it's message, the main thread's message can occur before the other threads finish printing.

b) Now, comment out pairs of sleep() statements as follows, and explain the results.

i. lines 35 & 37

```
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ gcc -pthread -o ex3 ex3.c -Wall
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ ./ex3
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Main thread says hi!
Thread Thread 1:1 says hi!
```

The time between the thread creation is zero. Therefore all 5 threads get created with the same id 1. The individual threads print the first 'Hi' message. There is no delay between the end of thread creation and the message of the main thread. Therefore it gets printed asynchronously. The exit() is reached in the main thread without any delay. None of the other threads make it to the second iteration before that.

ii. lines 12 & 35 (run the program multiple times, and explain changes in the results, if any)

```
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ gcc -pthread -o ex3 ex3.c -Wall
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ ./ex3
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 2:1 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Thread 3:1 says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 1:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!
Thread 1:1 says hi!
Thread 5:2 says hi!
Thread 5:3 says hi!
Main thread says hi!
```

Results change. But the last line printed is always by the main thread. Individual threads print asynchronously. The asynchronous running gives rise to race conditions, the results change because of this .

iii. lines 12 & 37 (increase the sleep time of line 37 gradually to 5)

```
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ gcc -pthread -o ex3 ex3.c -Wall
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ ./ex3
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 2:1 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Thread 3:1 says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 4:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!
Thread 5:1 says hi!
Thread 5:2 says hi!
Thread 5:3 says hi!
Main thread says hi!
```

37 is already commented and therefore we cannot increase it. The code was tested while changing the line 35 from sleep(1) to sleep(5) There is no difference in the output. The output will change only if the time for creating a single thread and executing it is more than 1s. But the computer takes way less time than this.

c) Now, uncomment all sleep() statements.

i. Can you explain the results?

```
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ gcc -pthread -o ex3 ex3.c -Wall
sonali@sonali-X540UA:/media/sonali/D0A0A5FAA0A5E766/Semester6/CO327 Operating Sy
stems/lab03$ ./ex3
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:1 says hi!
Thread 1:3 says hi!
Thread 1:2 says hi!
Thread 1:1 says hi!
Thread 1:3 says hi!
Thread 1:2 says hi!
Thread 2:1 says hi!
Thread 2:3 says hi!
Thread 3:2 says hi!
Thread 3:1 says hi!
Thread 4:3 says hi!
Thread 4:2 says hi!
Main thread says hi!
Thread 5:3 says hi!
Thread 5:3 says hi!
```

The threads are asynchronous because they are not connected by join() blocking. More than one thread may start with the same thread count variable. The thread's count might change while they are in the middle of their internal iterations. Therefore, 5.2 can occur without 5.1. The main thread may finish printing its message before the other threads complete since there is only 1s delay between the thread creation and main thread's printing.

ii. Does the output change if you revert the sleep value in line 37 back to 1? If so, why?

Yes. Now consistently the main thread's message comes in the last line.

3. Consider the following statement: “you use sleep() statements instead of join calls to get the desired output of a multithreaded program.”

a) Write a short critique of this statement expressing your views and preferences, if any.

Sleep can be used to mimic almost anything that can be achieved by joining if we have an estimate for the portions of code to run (including the time needed to start new threads, perform IO operations etc:). It is impossible to estimate these time periods with 100% certainty. Reducing uncertainty itself requires introducing higher sleep() times. But this is a huge waste of time. The same output can be achieved in a short time with join.

Exercise 4

1.Use the following skeleton code to implement a multi-threaded server.

ex4.c

2. Why do you need to declare connfd as a variable in the heap? What problem might occur if it was declared as a local variable?

If you use a local variable, it will be initialized in the main thread's stack. It would be safe to pass the address of that variable to respective threads as the main thread's stack variables lifetime will obviously be long enough. But the problem in doing so is you are updating the fd value everytime in while(1) in the same local variable which will make every thread to use that resulting in undefined behavior.
You will have to allocate new variable either in heap or stack for each thread for this reason so that each thread should be able to read correct fd value without any ambiguity