

Page 1:

1	2	3
✓	✓	✓
4	5	6
✓	✓	✓
7	8	9
✓	✓	✓
10	11	
✓	✓	

[Quiz Information](#)

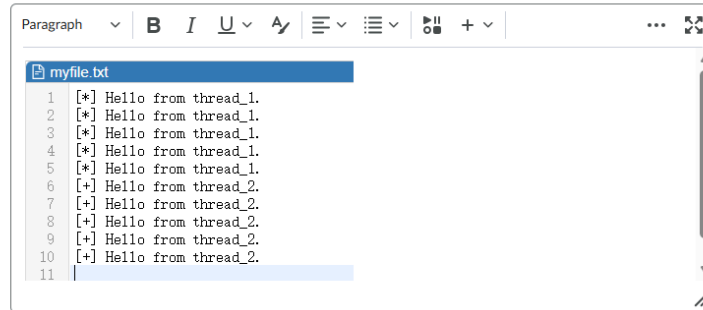
Add a File

Record Audio

Record Video

Question 2 (1 point) ✓ *Saved*

After running the program, insert a screenshot showing the contents of myfile.txt. You may need to change the file's permissions using a command like `chmod 444 myfile.txt`.



```
1 [*] Hello from thread_1.
2 [*] Hello from thread_1.
3 [*] Hello from thread_1.
4 [*] Hello from thread_1.
5 [*] Hello from thread_1.
6 [+] Hello from thread_2.
7 [+] Hello from thread_2.
8 [+] Hello from thread_2.
9 [+] Hello from thread_2.
10 [+] Hello from thread_2.
11
```

Add a File

Record Audio

Record Video

Question 3 (2 points) ✓ *Saved*

Based on the output of your code, do you think that the global variable "counter" is shared between threads 1 and 2 (Y/N)?

From the contents of myfile.txt, is the file myfile.txt shared between the two threads (Y/N)?

Page 1:

1 ✓	2 ✓	3 ✓
4 ✓	5 ✓	6 ✓
7 ✓	8 ✓	9 ✓
10 ✓	11 ✓	

Quiz Information

Question 4 (1 point) ✓ Saved

Change the sleep time from 1 to 100, then recompile your program and run it using

```
./main1.out & pstree -p | grep main1.out
```

From the output, Obtain the *pid* and *tid* for the process and threads. Insert a screenshot of your output.

Add a File

Record Audio

Record Video

Question 5 (1 point) ✓ Saved

After running (make sure the sleep time is already changed from 1 to 100)

```
./main1.out & pstree -p | grep main1.out
```

In the second terminal use **kill -9 <tid>** to terminate one of the threads. Does the main process remain alive after the thread is killed (Y/N)?

Y

Question 6 (1 point) ✓ Saved

If a process creates two threads, which of the following statements is true?

- ☐ Each thread receives a completely new PID and no TID
- ☒ Both threads share the same PID but have different TIDs
- ☐ Both threads have different PIDs but identical TIDs

Q7

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ucontext.h>
#include <sys/syscall.h>
```

```
#define STACK_SIZE 8192
ucontext_t main_ctx, th1_ctx, th2_ctx;
```

```
void thread_1() {
```

```

int c = 0;
while(1){
    c++;
    printf("Thread 1.\n");
    if (c%5==0){
        sleep(1);
        swapcontext(&th1_ctx, &th2_ctx);
    }
}
}

```

```

void thread_2() {
    int c = 0;
    while(1){
        c++;
        printf("Thread 2.\n");
        if (c%3==0){
            sleep(1);
            swapcontext(&th2_ctx, &th1_ctx);
        }
    }
}

```

```

int main() {
    // Allocate stacks
    char *sum_stack = malloc(STACK_SIZE);
    char *sort_stack = malloc(STACK_SIZE);

    if (!sum_stack || !sort_stack) {
        perror("malloc");
        exit(1);
    }

    // Create sum thread context
    getcontext(&th1_ctx);
    th1_ctx.uc_link = 0;
    th1_ctx.uc_stack.ss_sp = sum_stack;

```

```
th1_ctx.uc_stack.ss_size = STACK_SIZE;  
makecontext(&th1_ctx, thread_1, 0);
```

```
// Create sort thread context
```

```
getcontext(&th2_ctx);  
th2_ctx.uc_link = 0;  
th2_ctx.uc_stack.ss_sp = sort_stack;  
th2_ctx.uc_stack.ss_size = STACK_SIZE;  
makecontext(&th2_ctx, thread_2, 0);
```

```
setcontext(&th1_ctx);
```

```
free(sum_stack);  
free(sort_stack);  
return 0;
```

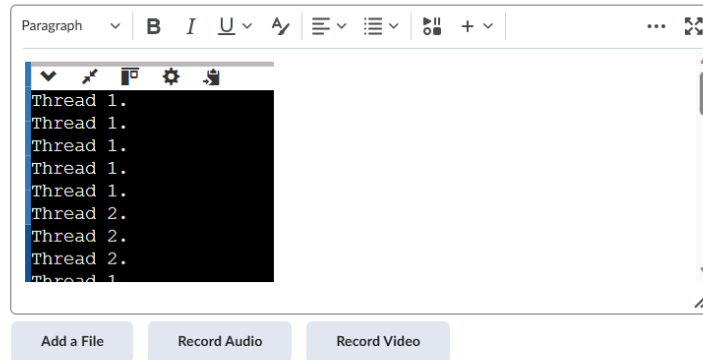
```
}
```

Page 1:

1	2	3
✓	✓	✓
4	5	6
✓	✓	✓
7	8	9
✓	✓	✓
10	11	
✓	✓	

Quiz Information

Insert a screenshot of your output.



Question 8 (2 points) ✓ Saved

Which thread starts first? e.g. 1 or 2

Modify thread_1() and thread_2() to print the thread ID using

```
printf("Thread 1.\n"); ➔ printf("Thread 1. PID: %d TID: %d\n", getpid(), syscall(SYS_gettid));  
printf("Thread 2.\n"); ➔ printf("Thread 2. PID: %d TID: %d\n", getpid(), syscall(SYS_gettid));
```

Does each thread have a different thread ID (Y/N)?

Change the sleep(1) calls to sleep(100) in both thread functions.

Does one sleep call block both threads (Y/N)? Does one sleep call block the main process (Y/N)?

Question 9 (1 point) ✓ Saved

A process uses two user-level threads managed entirely in user space. Which of the following statements is true regarding the PID and TIDs seen by the Linux kernel?

- ☐ The kernel assigns a different PID to each user-level thread
- ☐ Each user-level thread has a unique TID, but they all share the same PID
- ☒ The process has one PID, and both user-level threads share the same TID as the PID

Question 10 (1 point) ✓ Saved

Use the online C compiler

https://www.onlinegdb.com/online_c_compiler

to run your code and then answer the following questions.

```
1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <ctype.h>
6 #include <sys/wait.h>
7
8 int global_var = 0;
9
10 int main(){
11     int status;
12     int pid_t, fork_return;
13     fork_return = fork();
14
15     if (fork_return==0){
16         global_var++;
17         printf("Child pid = %d \t global_var = %d\n", getpid(), global_var);
18     }else{
19         global_var++;
20         waitpid(-1, &status, 0);
21         printf("Parent pid = %d \t global_var = %d\n", getpid(), global_var);
22     }
23     return 0;
24 }
```

Insert a screenshot of your output.



Page 1:

1 ✓	2 ✓	3 ✓
4 ✓	5 ✓	6 ✓
7 ✓	8 ✓	9 ✓
10 ✓	11 ✓	

Quiz Information

```
23     return 0;  
24 }
```

Insert a screenshot of your output.

Paragraph

B *I* U

```
Child pid = 1237    global_var = 1  
Parent pid = 1233   global_var = 1
```

Add a File

Record Audio

Record Video

Question 11 (1.5 points) ✓ Saved

From the output, determine the value of the `global_var` variable in the parent process. e.g. 2

From the output, determine the value of the `global_var` variable in the child process. e.g. 2

Based on your observations, do you think that the `global_var` variable is shared between the parent and child processes (Y/N)?

Submit Quiz

11 of 11 questions saved

