

The first program implements a new thread using `p_thread`. This program demonstrates, to my surprise, that the new thread shares a process id with the new thread but does have a new thread id.

```
//Brandy Poag-Dorado

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <sys/stat.h>
#include <stdlib.h>
#include <pthread.h>

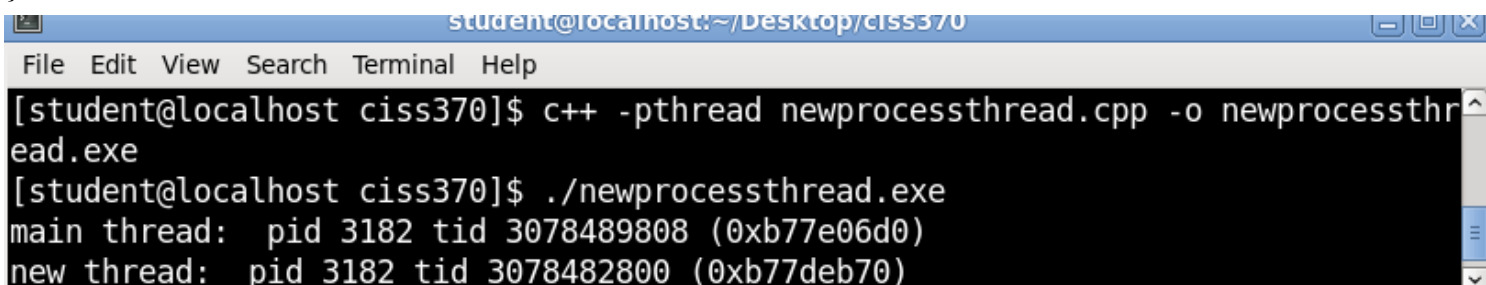
pthread_t n_tid;

void printids(const char *s)
{
    pid_t pid = getpid();
    pthread_t tid = pthread_self();
    printf("%s pid %u tid %u (0x%x)\n", s, (unsigned int)pid, (unsigned int)tid, (unsigned int)tid);
}

void * thr_fn(void *arg)
{
    printids("new thread: ");
    return((void *)0);
}

int main(int argc, char * argv[])
{
    int err = pthread_create(&n_tid, NULL, thr_fn, NULL);
    if (err != 0)
    {
        fprintf(stderr, "can't create thread: \n");
        exit(0);
    }
    printids("main thread: ");
    sleep (1);
    exit(0);

    return 0;
}
```



```
student@localhost: ~/Desktop/ciss370
File Edit View Search Terminal Help
[student@localhost ciss370]$ c++ -pthread newprocessthread.cpp -o newprocessthread.exe
[student@localhost ciss370]$ ./newprocessthread.exe
main thread:  pid 3182 tid 3078489808 (0xb77e06d0)
new thread:   pid 3182 tid 3078482800 (0xb77deb70)
```

The second program implements a thread using clone. This program demonstrates that the memory space is shared by declaring a global variable and updating the variable in the new thread afterwards the main thread accesses the global and we see that it is changed. The amount of sharing taking place depends on the flags specifying the things to share.

At least that is what I thought but now I see they have the different pid's and the same tid's?

```
//Brandy Poag-Dorado
```

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <sys/stat.h>
#include <stdlib.h>
#include <pthread.h>
#include <sched.h>
#include <sys/wait.h>
#include <signal.h>
#include <malloc.h>
```

```
int GLOBAL = 1000;
```

```
// The child thread will execute this function
```

```
int thdfunction( void* argument )
```

```
{
    pid_t pid = getpid();
    pthread_t tid = pthread_self();
    printf("In new thread pid %u tid %u (0x%x)\n", (unsigned int)pid,
(unsigned int)tid, (unsigned int)tid);
    GLOBAL += 5;
    printf( "In thread global is %i\n", GLOBAL);
    return 0;
}
```

```
int main()
```

```

{
    void* stack;

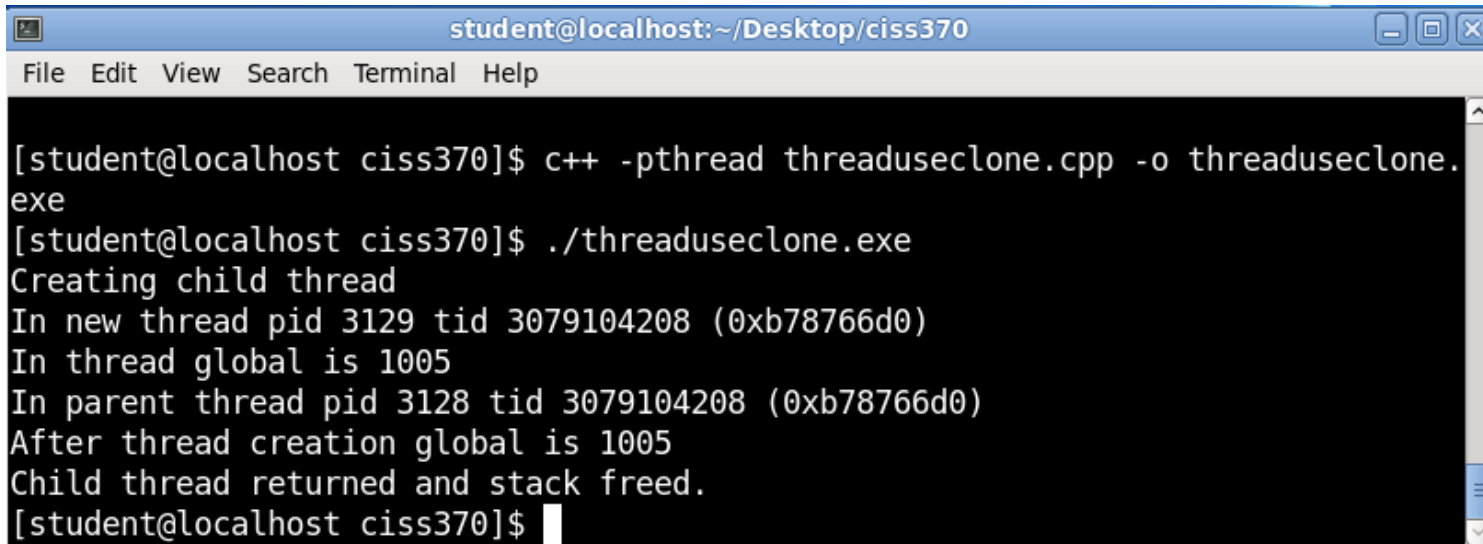
    // Allocate the stack
    stack = malloc(1024);
    if ( stack == 0 )
    {
        perror( "malloc: could not allocate stack" );
        exit( 1 );
    }
    printf( "Creating child thread\n" );

    // Call the clone system call to create the child thread
    pid_t pid = clone( &thdfunction, (char*) stack + 2000,
        SIGCHLD | CLONE_FS | CLONE_FILES | CLONE_SIGHAND |
        CLONE_VM, 0 );
    if ( pid < 0 )
    {
        perror( "clone" );
        exit( 2 );
    }

    // Wait for the child thread to exit
    pid = waitpid( pid, 0, 0 );
    if ( pid < 0 )
    {
        perror( "waitpid" );
        exit( 3 );
    }
    pid_t pid2 = getpid();
    pthread_t tid = pthread_self();
    printf("In parent thread pid %u tid %u (0x%x)\n", (unsigned int)pid2,
(unsigned int)tid, (unsigned int)tid);
    printf( "After thread creation global is %i\n", GLOBAL);
    // Free the stack
    free( stack );
}

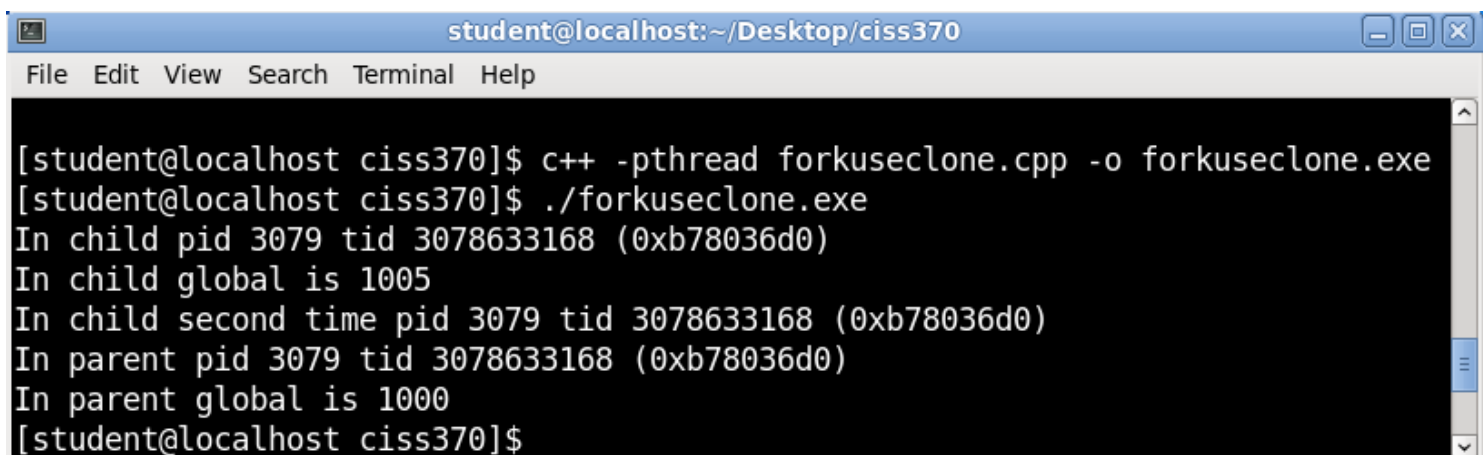
```

```
printf( "Child thread returned and stack freed.\n" );  
  
return 0;  
}
```



A terminal window titled "student@localhost:~/Desktop/ciss370" with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the compilation and execution of a C++ program using pthreads. The output indicates successful thread creation and stack freeing.

```
student@localhost:~/Desktop/ciss370  
File Edit View Search Terminal Help  
[student@localhost ciss370]$ c++ -pthread threaduseclone.cpp -o threaduseclone.exe  
[student@localhost ciss370]$ ./threaduseclone.exe  
Creating child thread  
In new thread pid 3129 tid 3079104208 (0xb78766d0)  
In thread global is 1005  
In parent thread pid 3128 tid 3079104208 (0xb78766d0)  
After thread creation global is 1005  
Child thread returned and stack freed.  
[student@localhost ciss370]$
```



A terminal window titled "student@localhost:~/Desktop/ciss370" with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the compilation and execution of a C++ program using pthreads. The output indicates successful thread creation and stack freeing.

```
student@localhost:~/Desktop/ciss370  
File Edit View Search Terminal Help  
[student@localhost ciss370]$ c++ -pthread forkuseclone.cpp -o forkuseclone.exe  
[student@localhost ciss370]$ ./forkuseclone.exe  
In child pid 3079 tid 3078633168 (0xb78036d0)  
In child global is 1005  
In child second time pid 3079 tid 3078633168 (0xb78036d0)  
In parent pid 3079 tid 3078633168 (0xb78036d0)  
In parent global is 1000  
[student@localhost ciss370]$
```

uses, that creates a fork without using fork directly. This program demonstrates that the memory space is not shared by declaring a global variable and updating the variable in the child process afterwards the parent accesses its copy of the global and we see that it is unchanged.

At least that is what I thought but now I see they have the same pid's and the same tid's?

```
//Brandy Poag-Dorado
```

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <sys/stat.h>
#include <stdlib.h>
#include <pthread.h>
#include <sched.h>
#include <sys/wait.h>
#include <signal.h>
#include <malloc.h>
#include <syscall.h>
```

```
int GLOBAL = 1000;
```

```
int main()
{
    // Call the clone system call to create the fork using syscall
    pid_t pid = syscall(SYS_clone, CLONE_IO | SIGCHLD,
0,NULL,NULL);
    if ( pid < 0 )
    {
        perror( "clone" );
        exit( 2 );
    }
    else if (pid == 0) //child
    {
```

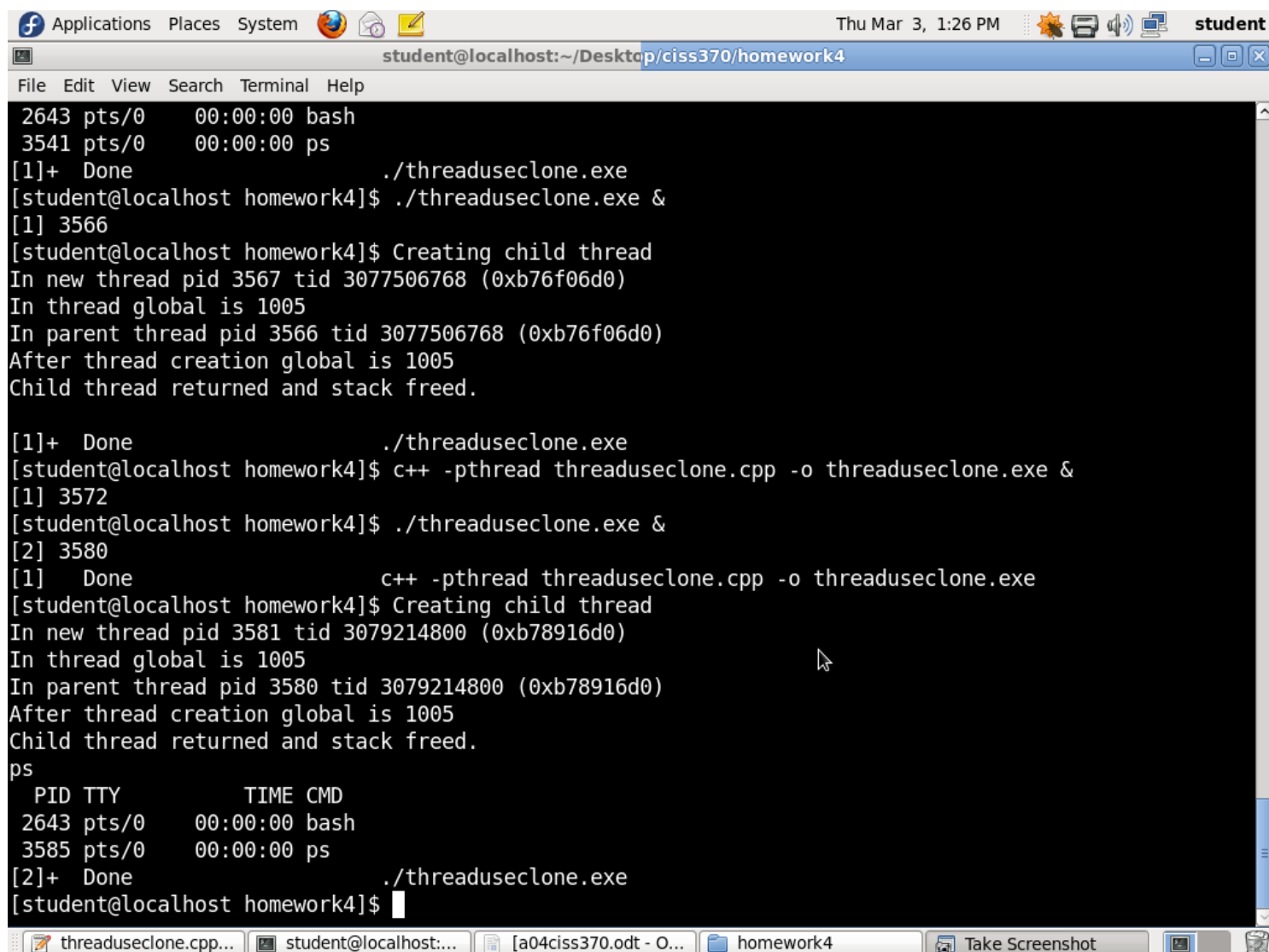
```

        pid_t pid = getpid();
        pthread_t tid = pthread_self();
        printf("In child pid %u tid %u (0x%x)\n", (unsigned int)pid,
(unsigned int)tid, (unsigned int)tid);
        GLOBAL += 5;
        printf( "In child global is %i\n", GLOBAL);
        pid = getpid();
        tid = pthread_self();
        printf("In child second time pid %u tid %u (0x%x)\n",
                (unsigned int)pid, (unsigned int)tid, (unsigned int)tid);
    }
    else //parent
    {
        if (wait(NULL) < 0)
        {
            perror("Error wait.");
        }
        pid_t pid = getpid();
        pthread_t tid = pthread_self();
        printf("In parent pid %u tid %u (0x%x)\n", (unsigned int)pid,
(unsigned int)tid, (unsigned int)tid);
        printf( "In parent global is %i\n", GLOBAL);
    }

    return 0;
}

```

Finally, the forth task is not possible. We can not kill a single thread within a process with multiple threads. The new thread shares a process id with the parent thread thus we can only kill the process therefore killing all threads of that process when you use the pthread to create the thread. If you use clone the pid's are different but the tid's are the same, however, I could not kill a process?



The screenshot shows a terminal window titled "student@localhost: ~/Desktop/ciss370/homework4". The terminal output is as follows:

```
2643 pts/0    00:00:00 bash
3541 pts/0    00:00:00 ps
[1]+  Done                  ./threaduseclone.exe
[student@localhost homework4]$ ./threaduseclone.exe &
[1] 3566
[student@localhost homework4]$ Creating child thread
In new thread pid 3567 tid 3077506768 (0xb76f06d0)
In thread global is 1005
In parent thread pid 3566 tid 3077506768 (0xb76f06d0)
After thread creation global is 1005
Child thread returned and stack freed.

[1]+  Done                  ./threaduseclone.exe
[student@localhost homework4]$ c++ -pthread threaduseclone.cpp -o threaduseclone.exe &
[1] 3572
[student@localhost homework4]$ ./threaduseclone.exe &
[2] 3580
[1]  Done                  c++ -pthread threaduseclone.cpp -o threaduseclone.exe
[student@localhost homework4]$ Creating child thread
In new thread pid 3581 tid 3079214800 (0xb78916d0)
In thread global is 1005
In parent thread pid 3580 tid 3079214800 (0xb78916d0)
After thread creation global is 1005
Child thread returned and stack freed.
ps
  PID TTY          TIME CMD
 2643 pts/0    00:00:00 bash
 3585 pts/0    00:00:00 ps
[2]+  Done                  ./threaduseclone.exe
[student@localhost homework4]$
```

The terminal window includes a menu bar (File, Edit, View, Search, Terminal, Help) and a taskbar at the bottom with open files: "threaduseclone.cpp...", "student@localhost:...", "[a04ciss370.odt - O...", "homework4", and a "Take Screenshot" button.