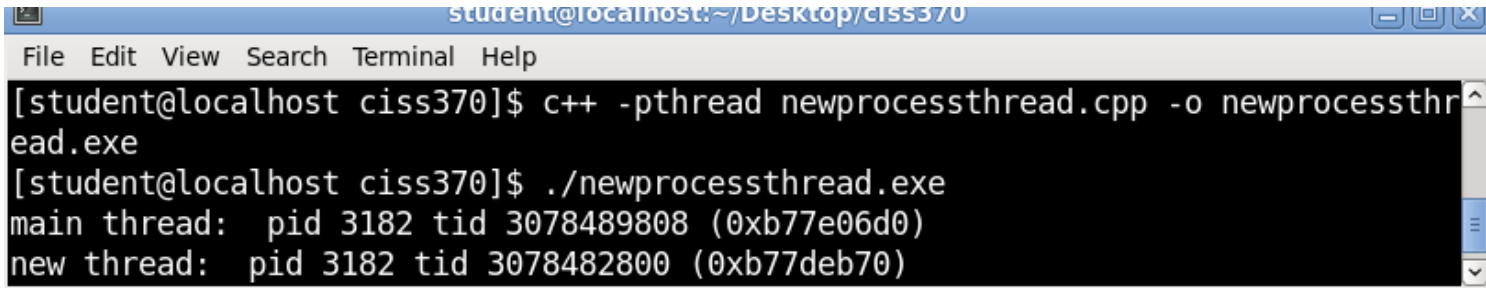
The first program implements a new thread using p_thread. This program demonstrates, to my surprise, that the new thread shares a process id with the new thread but does have a new thread id.

```
//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;
}
```

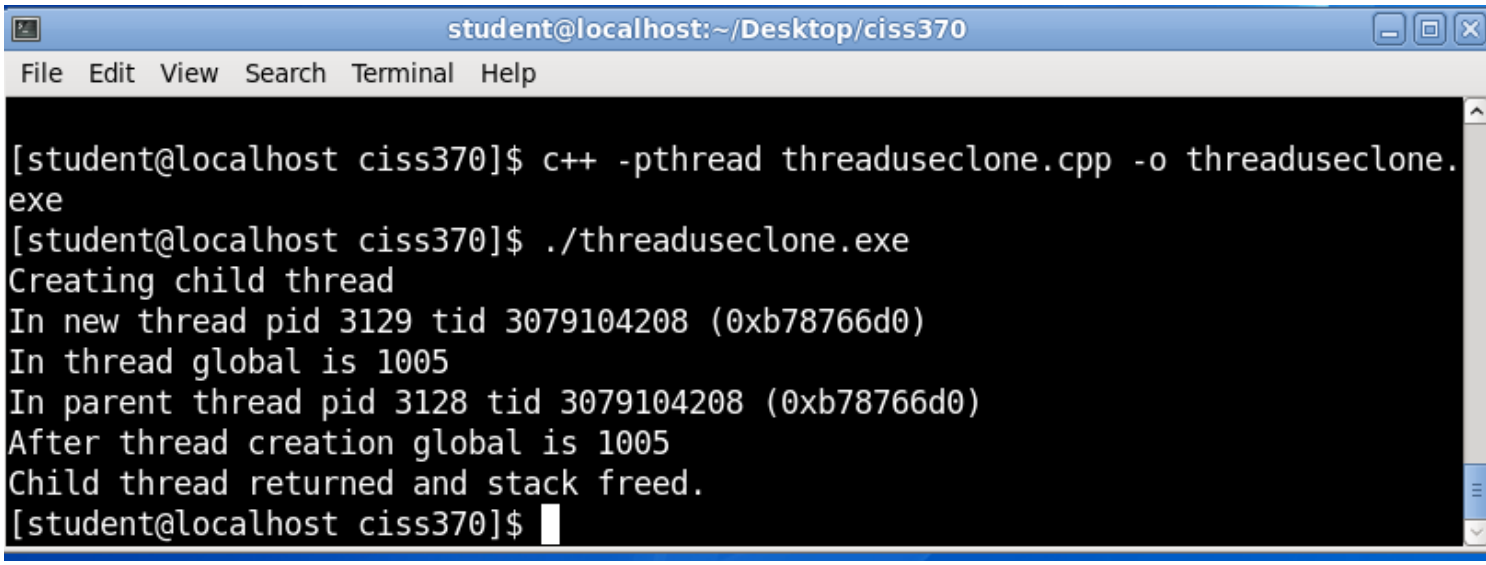


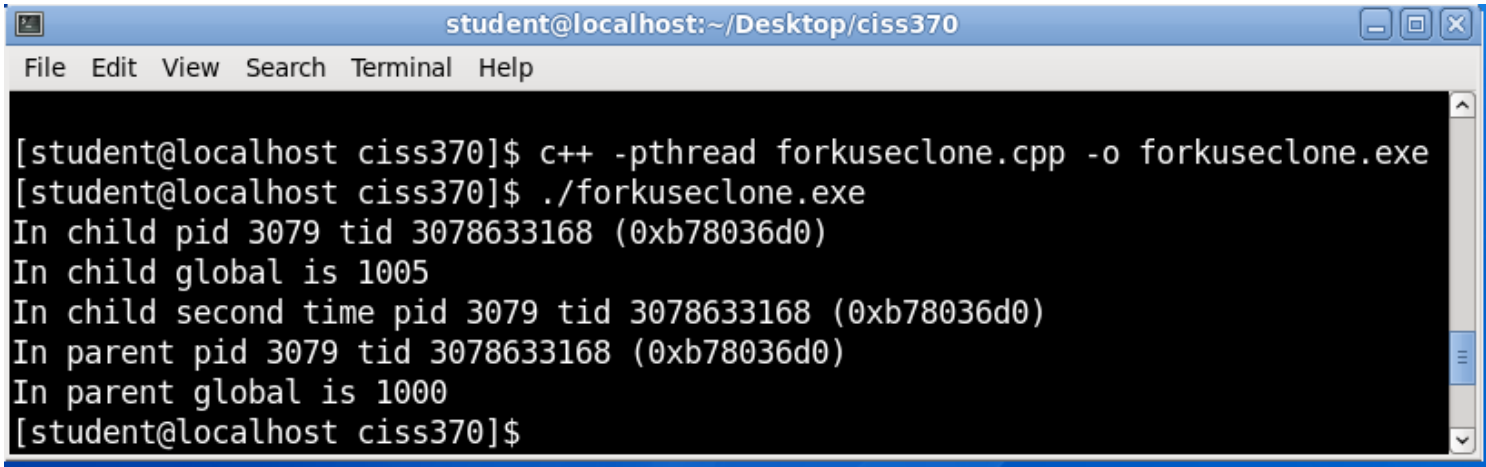
The second program implements a thread using clone. This program demonstrates that the memory space is shared by declaring a global variable and updating the variable in the new thread afterwards the main thread accesses the global and we see that it is changed. The amount of sharing taking place depends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





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, howener, I could not kill a process?