

threads & forks

Links: [COSC 3360](#)

Threads

- include `pthread.h` file to use pthreads
- use `pthread_create(&threadname, NULL, threadingFunctionName, (void*)threadFunArgs)` to create the thread
 - first argument: name of thread variable passed by reference
 - second argument: `NULL`
 - third argument: name of threading function
 - fourth argument: arguments of the threading function cast as a void pointer, put `NULL` if no arguments need to be passed in. If multiple parameters need to be passed in the threading function, make a struct with all the values and pass in a pointer to the struct (cast as void pointer) as the fourth argument
- use `pthread_join(threadName, NULL)` to "join" the threads (i.e. to wait for all the threads to complete)

the threading function has a return type of void pointer and returns `NULL`

If creating multiple threads in a loop using a dynamic array, make sure that the *pthread_join* occurs **outside** of the loop it was created in.

when creating multiple threads using dynamic array, have a dynamic array of threads and a dynamic array of pointers to your arguments struct, and make sure this is *static*

Example

```

#include <iostream>
#include <pthread.h>
#include <vector>
#include <string>

using namespace std;

#define NUM_THREADS 5

struct thread_data {
    int thread_id;
    string message;
};

void *PrintHello(void *threadarg) {
    thread_data *my_data = (thread_data *) threadarg;
    string temp = my_data->message + " from thread with id: " +
to_string(my_data->thread_id);
    my_data->message = temp;

    return NULL;
}

int main () {
    vector<pthread_t> threads;
    static vector<thread_data*> threadArgs;

    for(int i = 0; i < NUM_THREADS; i++ ) {
        thread_data* td = new thread_data();
        td->thread_id = i;
        td->message = "This is a message";

        pthread_t thread;

        int rc = pthread_create(&thread, NULL, PrintHello, (void

```

```

*)td);

    if (rc) {
        cout << "Error:unable to create thread," << rc <<
endl;
        exit(-1);
    }

    threads.push_back(thread);
    threadArgs.push_back(td);
}

for(int i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
}

for(thread_data* messages: threadArgs) {
    cout << messages->message << endl;
}
}

```

Output

This is a message from thread with id: 0
 This is a message from thread with id: 1
 This is a message from thread with id: 2
 This is a message from thread with id: 3
 This is a message from thread with id: 4

Forks

- include `unistd.h` and `sys/wait.h`
- create a fork using `fork()` it returns two values:

- to the parent process it returns the pid of the process
- to the child it returns `0`, thus you can check if you are in a child process by checking if the return value of `fork()` is `0`.
- `fork` makes a copy and runs the next instruction in the code
- you call `wait(nullptr)` to avoid zombie processes (so, the parent waits for the child to end) and you use `_exit(0)` to terminate the child process.

To make sure the fork outputs the code in a particular order, call `wait(nullptr)` inside the loop you create a fork.

each child process (including grandchild processes) will need their own `wait()`

Handy trick: call the wait function in the same scope as the fork function, call the exit function in the code that the child process will execute.

Example

```
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>

using namespace std;

int main() {
    int pid;
    cout << "I am the parent process" << endl;
    for(int i=0; i<3; i++) {
        pid = fork();
        if(pid == 0) {
            cout << "I am the child process " << i << endl;
            if(i == 1) {
                pid = fork();
            }
        }
    }
}
```

```

        if(pid == 0) {
            cout << "I am grandchild process " << i <<
endl;

            _exit(0);
        }
        wait(nullptr);
    }
    _exit(0); //ends the child process
}
wait(nullptr); // <-- guarantees it will happen in
certain order
}
}

```

Output

```

I am the parent process
I am the child process 0
I am the child process 1
I am grandchild process 1
I am the child process 2

```