Computer Operating System Experiment

Threads

Lab 4

Roadmap

- What is the thread?
- How to use pthread in Linux?
- Q&A

Objective:

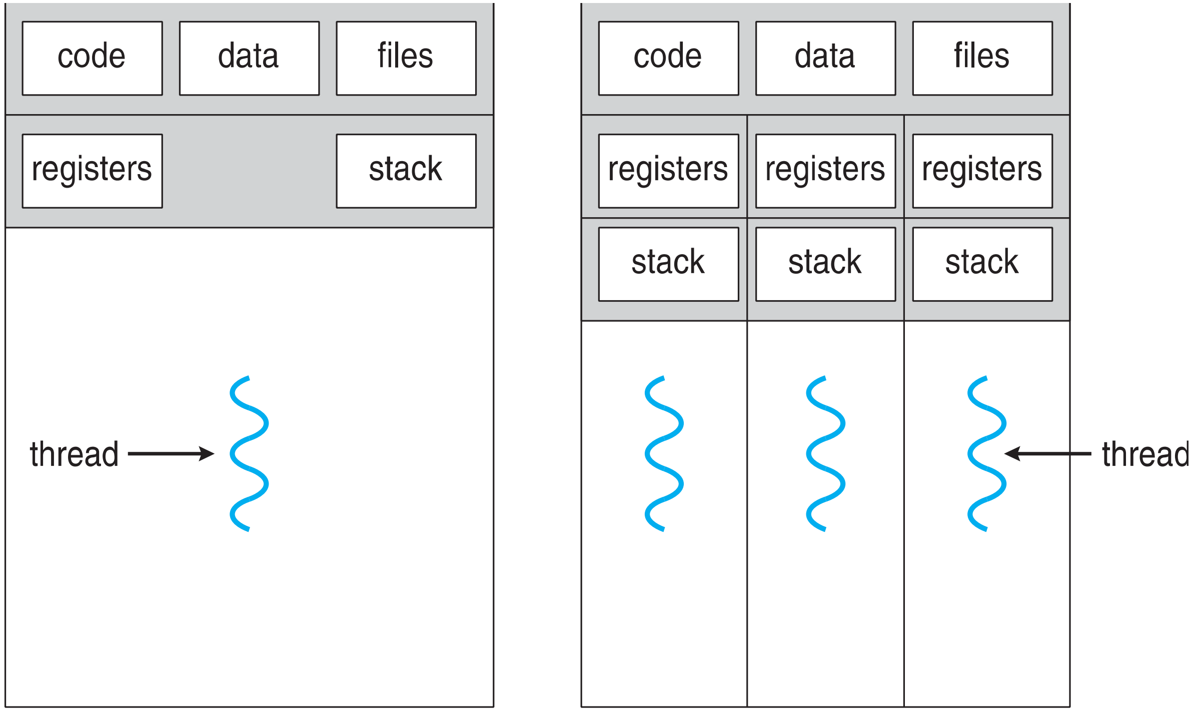
- Practice working with multi-threaded
- Data sharing mechanism
- Thread Parallelism

What are Threads? Threads vs Processes

- A Process is an instance of a computer program that is being executed.
- A Thread is an instance of a sequential computer program that is being executed.
  - Threads are the basic unit for scheduling in modern OS
  - A process contains at least one thread
  - A process may contain multiple threads for parallel execution
- Threads of the same processes share memory space; i.e., they accesses the same chunk of memory with the same address
  - Threading represents the OS support for shared-memory programming

What are threads?

- A basic unit of CPU utilization
  - Private: Thread ID, program counter, register set, stack
  - Shared: code section, data section, OS resources (IO & file)



single-threaded process

multithreaded process

What is Pthreads?

- POSIX Threads
- Defines standard threads API supported on almost all platforms
- Concepts behind Pthreads interface are broadly applicable
- Pthreads standard interfaces for users to use the OS threads of any vendors (as long as the vendor follows the standard)
  - Improved portability
  - Also helps OS vendors to properly implement their threads

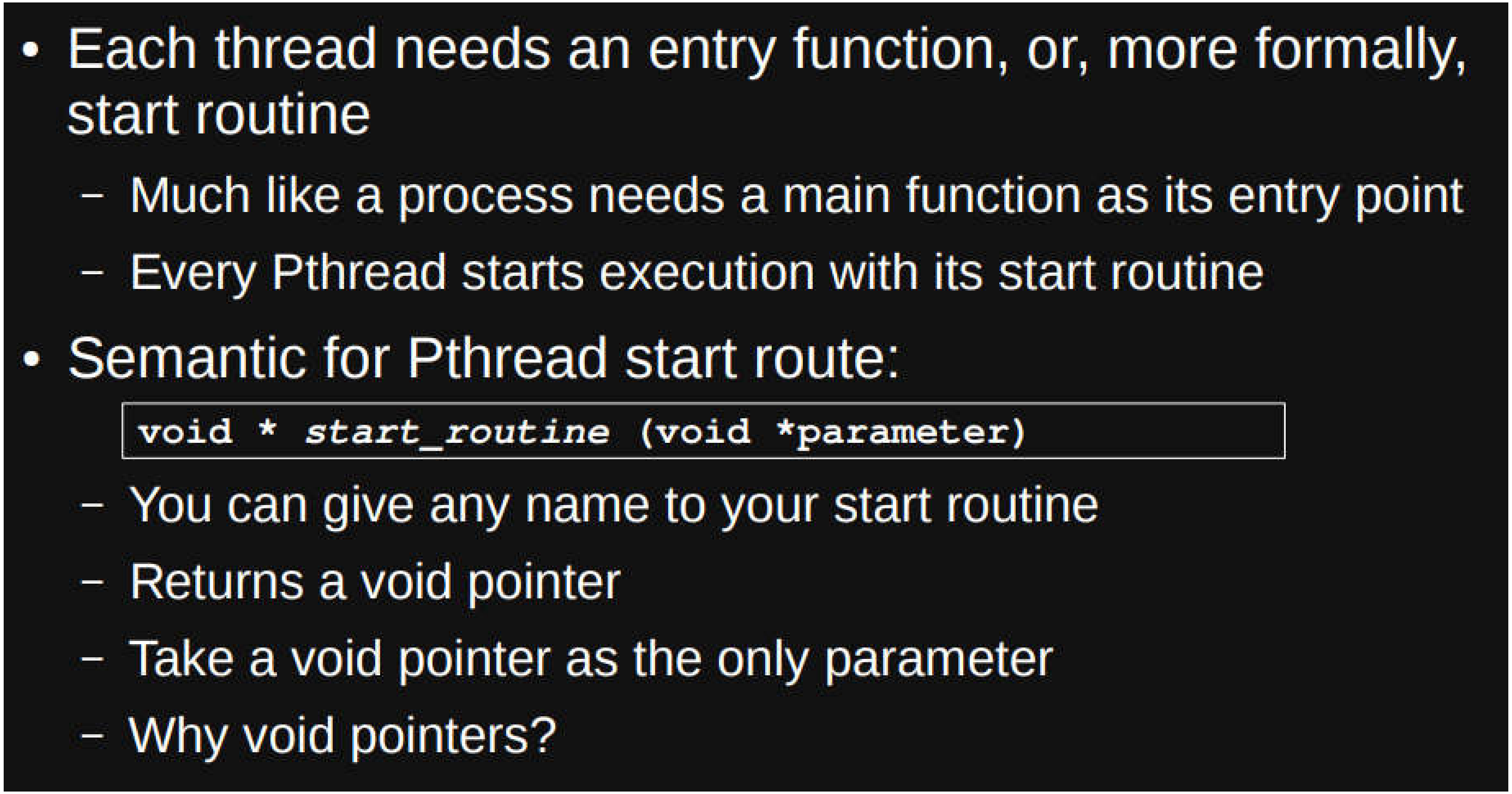
Compiling Pthreads

- On Linux, Pthreads are provided with library libpthreads. GCC also has additional supports for Pthreads compilation.
- Always include <pthread.h>
- To compile a C/C++ Pthread program, use command line option “-lpthread”
  - – Example: gcc -lpthread pthreads_code.c -o pthread_exec
- To execute a Pthread program, just run the executable normally.
  - But it is the Pthread program’s responsibility to create threads, determine thread count, and map tasks to the threads

Pthread APIs

- Thread execution management APIs
  - – functions start with “pthread_”
- Thread property (attributes) management APIs
  - – Functions start with “pthread_attr”
- Synchronization APIs
  - – Mutex: “pthread_mutex_” and “pthread_mutexattr_”
  - – Conditional variables: “pthread_cond_” and “pthread_condattr_”
- Thead specific data
  - Functions start with “pthread_key_”

Creating A New Thread

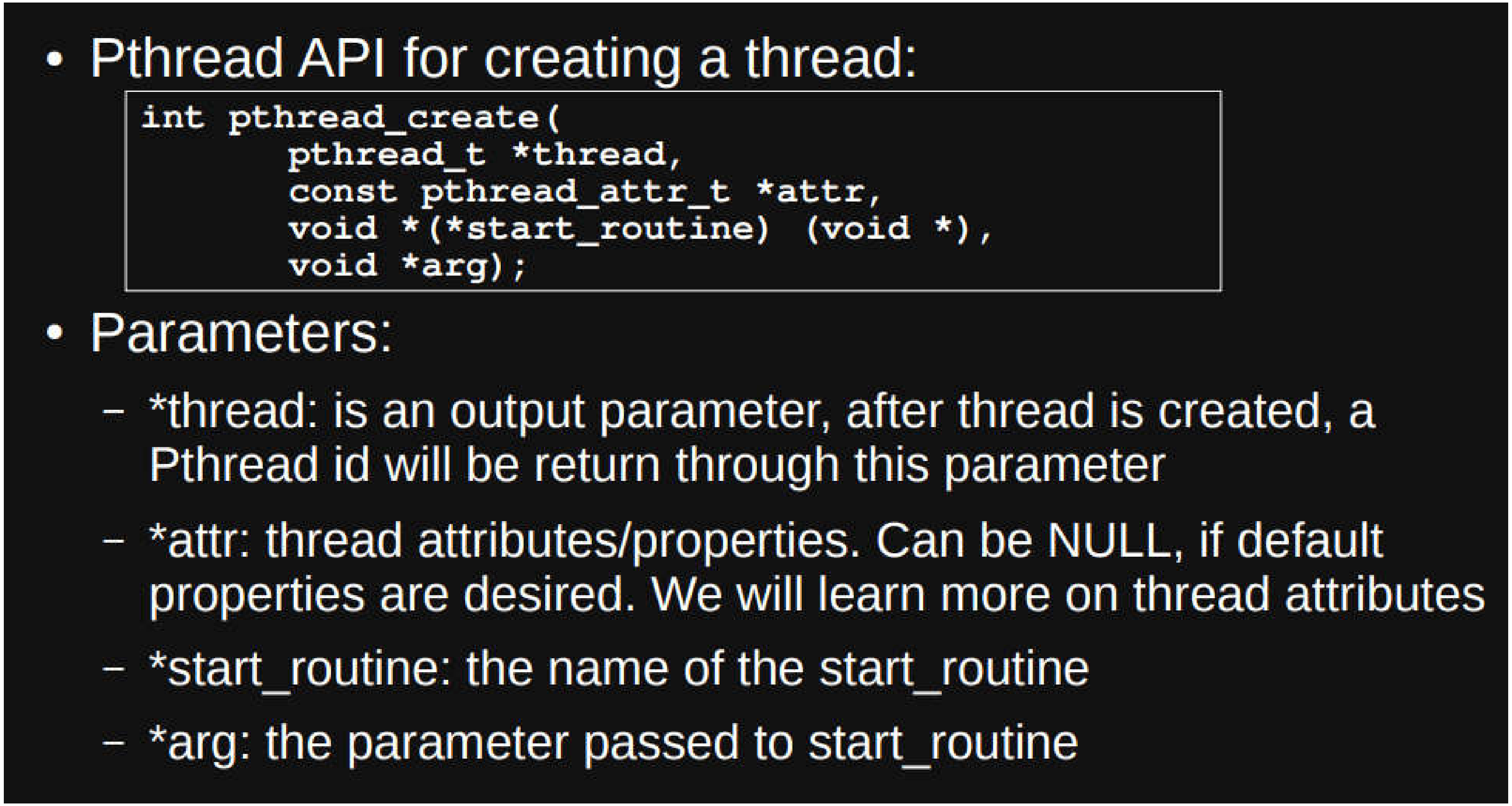


- Each thread needs an entry function, or, more formally, start routine
  - Much like a process needs a main function as its entry point
  - Every Pthread starts execution with its start routine
- Semantic for Pthread start route:

```
void * start_routine (void *parameter)
```

  - You can give any name to your start routine
  - Returns a void pointer
  - Take a void pointer as the only parameter
  - Why void pointers?

Creating A New Thread cont’d

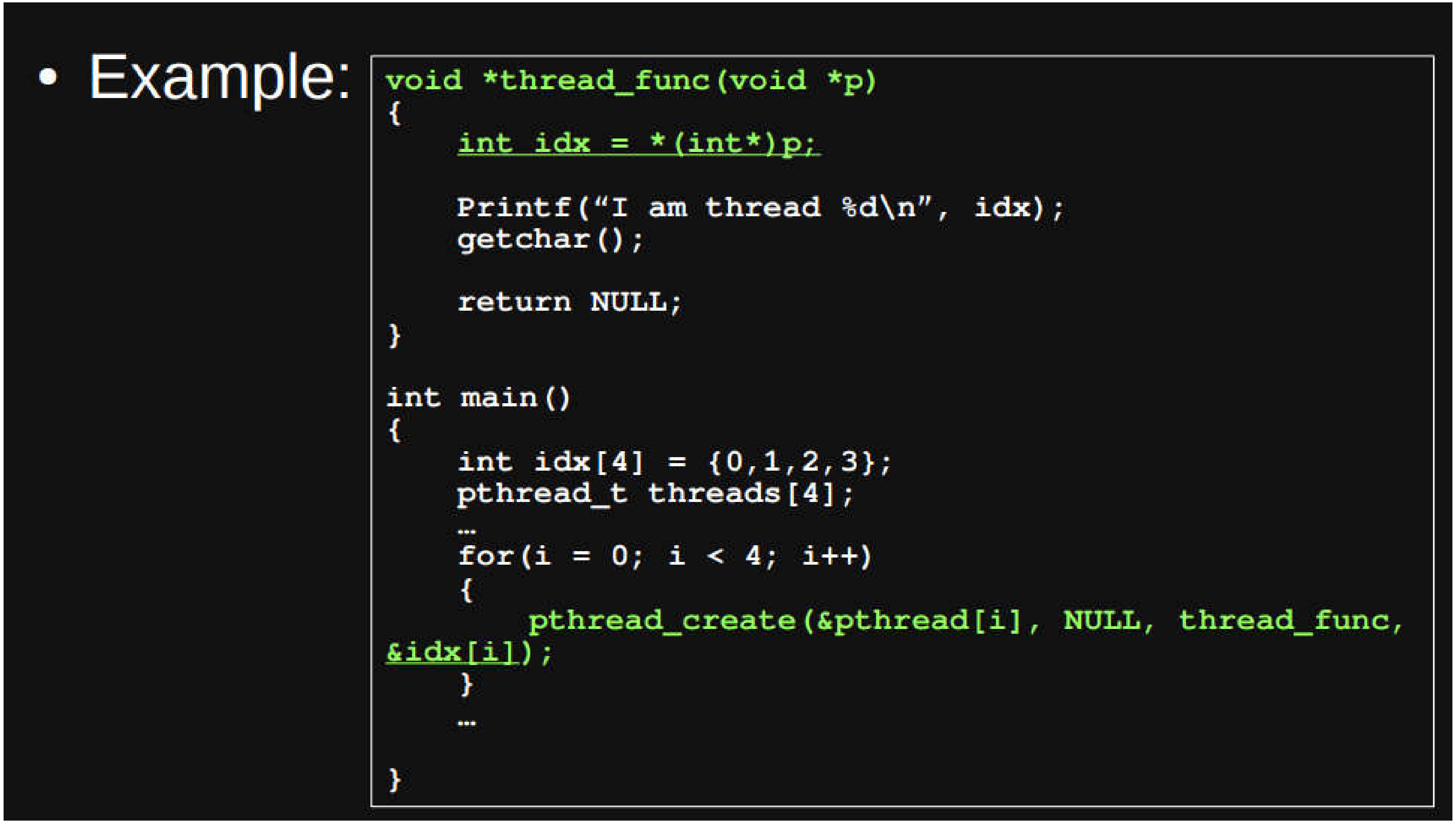


- Pthread API for creating a thread:

```
int pthread_create(
        pthread_t *thread,
        const pthread_attr_t *attr,
        void *(*start_routine) (void *),
        void *arg);
```

- Parameters:
  - *thread: is an output parameter, after thread is created, a Pthread id will be return through this parameter
  - *attr: thread attributes/properties. Can be NULL, if default properties are desired. We will learn more on thread attributes
  - *start_routine: the name of the start_routine
  - *arg: the parameter passed to start_routine

Creating A New Thread cont’d



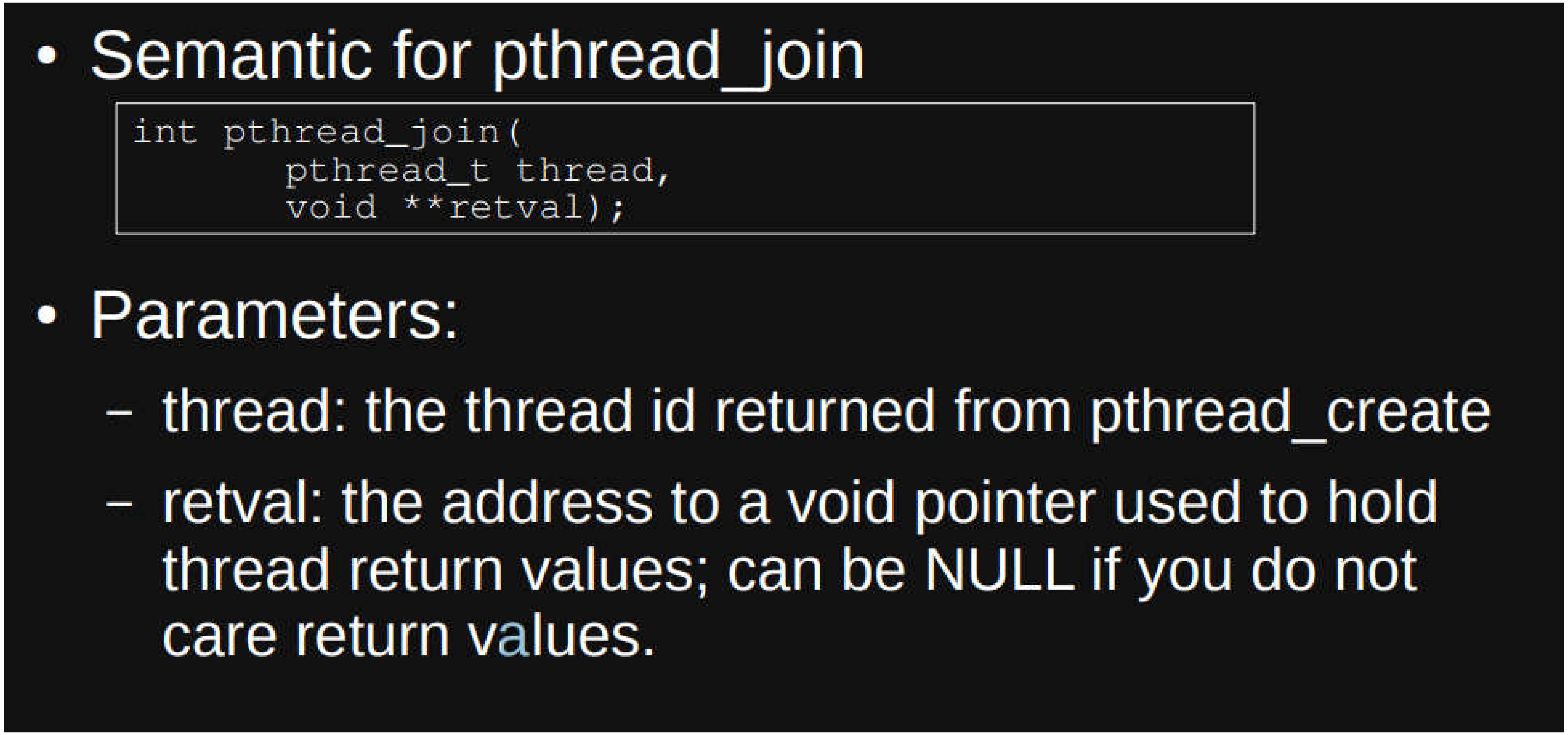
- Example:

```
void *thread_func(void *p)
{
    int idx = *(int*)p;

    Printf("I am thread %d\n", idx);
    getchar();

    return NULL;
}

int main()
{
    int idx[4] = {0,1,2,3};
    pthread_t threads[4];
    …
    for(i = 0; i < 4; i++)
    {
        pthread_create(&pthread[i], NULL, thread_func,
&idx[i]);
    }
    …

}
```

Waiting for A Thread to Finish

- Most of the time, the main thread of a process should only quit if all other threads have finished execution
  - – What happens if the main thread does not wait?
- Pthread provides a function pthread_join to allow a program to determine whether a thread has finished
- pthread_join also allows the retrieval of thread return values.

Waiting for A Thread to Finish cont'd



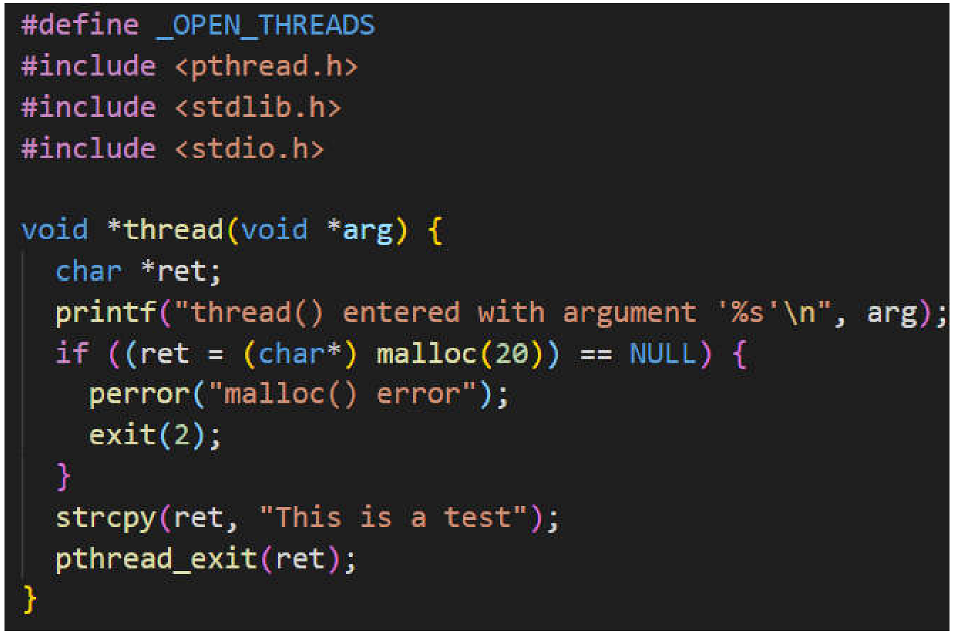
- Semantic for pthread_join

```
int pthread_join(
        pthread_t thread,
        void **retval);
```

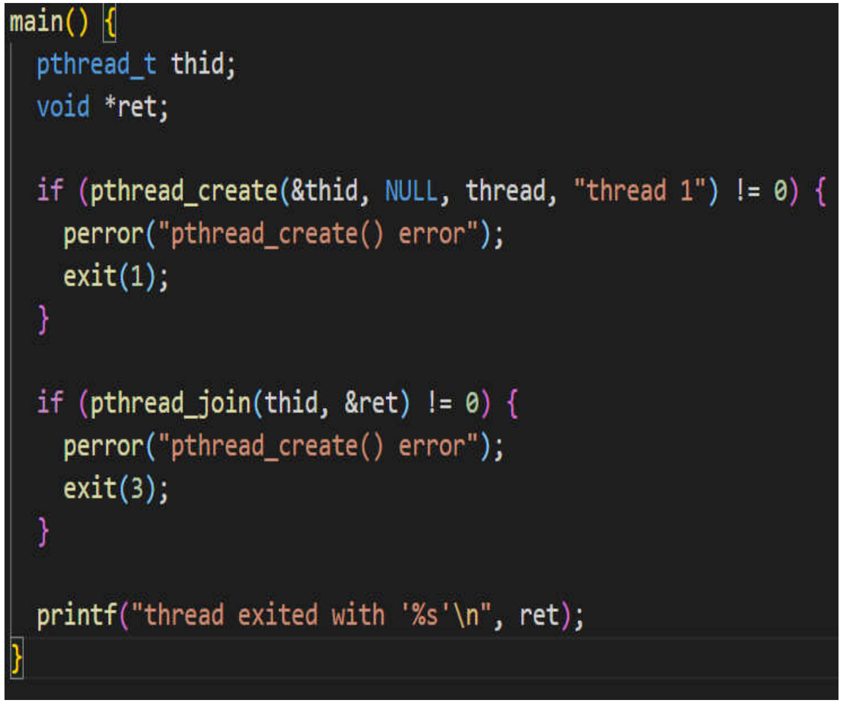
- Parameters:
  - thread: the thread id returned from pthread_create
  - retval: the address to a void pointer used to hold thread return values; can be NULL if you do not care return values.

Waiting for A Thread to Finish cont’d

Example：



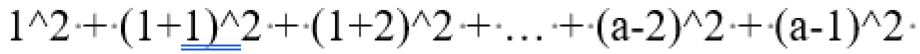
```c
#define _OPEN_THREADS
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

void *thread(void *arg) {
  char *ret;
  printf("thread() entered with argument '%s'\n", arg);
  if ((ret = (char*) malloc(20)) == NULL) {
    perror("malloc() error");
    exit(2);
  }
  strcpy(ret, "This is a test");
  pthread_exit(ret);
}
```



```c
main() {
  pthread_t thid;
  void *ret;

  if (pthread_create(&thid, NULL, thread, "thread 1") != 0) {
    perror("pthread_create() error");
    exit(1);
  }

  if (pthread_join(thid, &ret) != 0) {
    perror("pthread_create() error");
    exit(3);
  }

  printf("thread exited with '%s'\n", ret);
}
```

Experiments

- Experiment 1:
  - data sharing
- Experiment 2:
  - calculates various statistical values using pthread
- Experiment 3:
  - calculate the sum of number of squares



1^2 + (1+1)^2 + (1+2)^2 + … + (a-2)^2 + (a-1)^2

  - Implement a serial program first , then modify it to pthread version

Q&A