POLITECNICO
DI TORINO

Multithread Programming

Operating Systems – Sarah Azimi

Threads

- Threads / Processes are the mechanism which allows you to run two or more program concurrently (multitasking).
  - Process-based multitasking handles the concurrent executions of programs.
  - Thread-based multitasking deals with the concurrent execution of pieces of the same program.

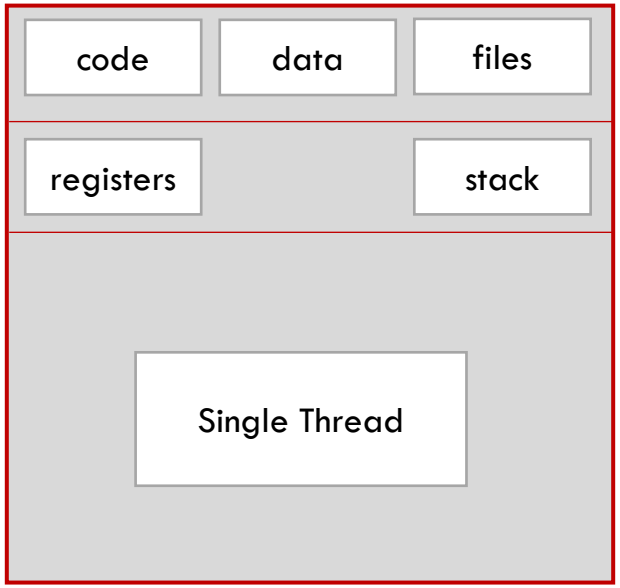
Threads

- Threads / Processes are the mechanism which allows you to run two or more program concurrently (multitasking).
  - Process-based multitasking handles the concurrent executions of programs.
  - Thread-based multitasking deals with the concurrent execution of pieces of the same program.

- Threads are lightweight
  - Creating a thread is more efficient than creating a process.
  - Communication between threads is easier than between processes.
  - Context switching between threads requires fewer CPU cycles and memory references than switching processes.
  - Threads only track a subset of process state (share list of open files, pid, …).

Threads

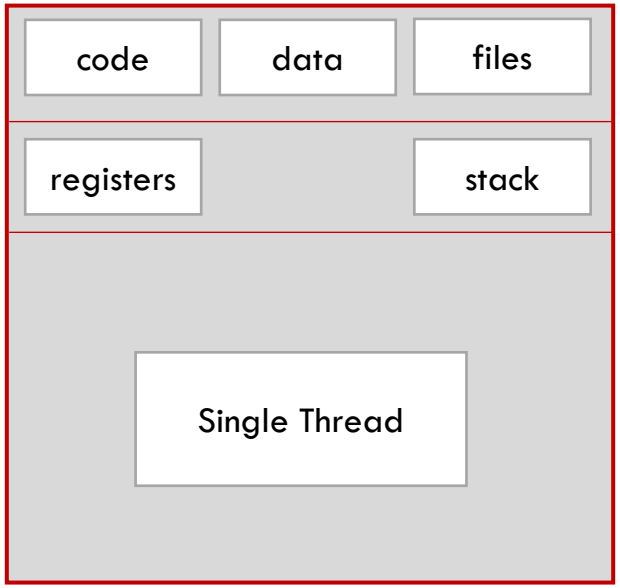
- In traditional operating systems, each process has an address space and a single thread of control.



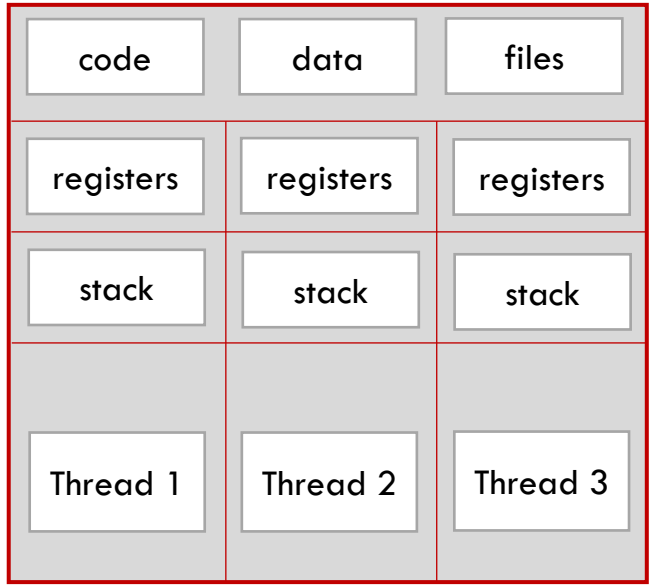
Single threaded process

Threads

- In traditional operating systems, each process has an address space and a single thread of control.
- There are frequently solutions in which it is desirable to have multiple threads of control in the same address space running concurrently, as though they are (almost) separate tasks.



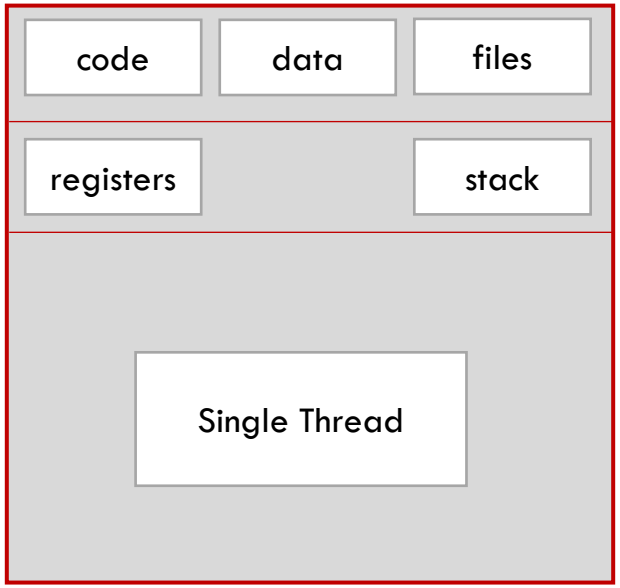
Single threaded process



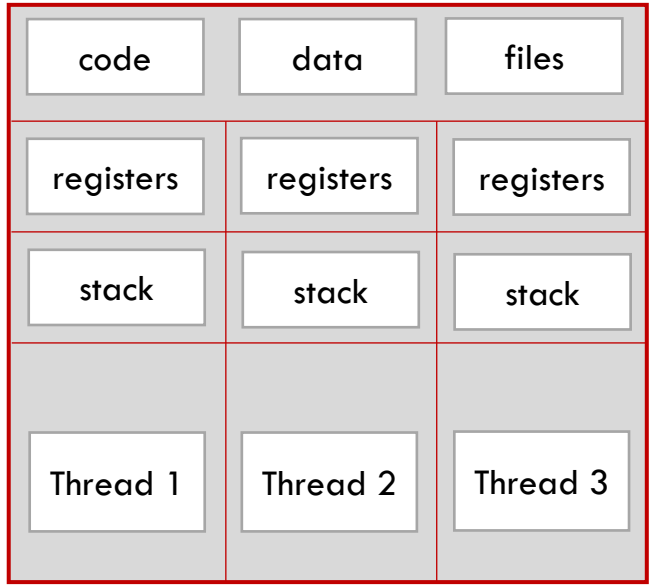
Multithreaded process

Threads vs. Processes

- Threads in the same process shared:
  - Process instructions
  - Most data
  - Open files
  - Signal and Signal handlers
  - Current working directory
- Each thread has a unique:
  - Thread Identifier
  - Program counter
  - Set of registers, stack pointer
  - Stack for local variables



Single threaded process



Multithreaded process

Threads

- Usage of A POSIX standard API to:
  - Create and destroy threads
  - Synchronization between thread
  - Data management
  - Thread scheduling

Threads

- Usage of A POSIX standard API to:
  - Create and destroy threads
  - Synchronization between thread
  - Data management
  - Thread scheduling

- pthread functions are degined in the header file `<pthread.h>`
  - `#include <pthread.h>`
- It is required to compile the source code with the option `-pthread` or including the library of pthread.
  - `gcc myprogram.c -o myprogram` **`-pthread`**
  - `gcc myprogram.c -o myprogram` **`-lpthread`**

Creating Threads

- To create a thread:

- Syntax:

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

Creating Threads

- To create a thread:

- Syntax:

  int pthread_create (pthread_t *thread_id, const pthread_attr_t *attr, void *(* myThreadFun) (void *), void *arg);

  - thread_id is a pointer to a variable which is used to store the thread id of a new created thread.

  - Example:

```
pthread_t thread_id;
pthread_Create (&thread_id, …);
```

Creating Threads

- To create a thread:

- Syntax:

```
int pthread_create  (pthread_t *thread_id, const pthread_attr_t *attr, void *(* myThreadFun) (void *), void *arg);
```

  - attr is a pointer to a thread attribute object which is used to set thread attributes, NULL can be used to create a thread with default arguments.

Creating Threads

- To create a thread:

- Syntax:

int pthread_create (pthread_t *thread_id, const pthread_attr_t *attr, void *(* myThreadFun) (void *), void *arg);

  - myThreadFun is a pointer to a thread function. This function contains the code segment which is executed by the thread.

  - Example:

```
void *myThreadFun{}


int main(){
    pthread_t thread_id;
    pthread_Create (&thread_id, NULL,myThreadFun, … );
```

Creating Threads

- To create a thread:

- Syntax:

```
int pthread_create  (pthread_t *thread_id, const pthread_attr_t *attr, void *(* myThreadFun) (void *), void *arg);
```

  - arg is a single argument for thread function. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

Creating Threads

- To create a thread:

- Syntax:

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

  - Return value:
    - If successful, it returns 0. Otherwise, it returns a nonzero error.

Example

```
#include <stdio.h>
#include <pthread.h>
Int main()  {
            pthread_t f2_thread, f1_thread, f3_thread; int i1=1,i2=2;
             void *f2(), *f1(),*f3();
             pthread_create(&f1_thread,NULL,f1,&i1);
             pthread_create(&f2_thread,NULL,f2,&i2);
             pthread_create(&f3_thread,NULL,f3,NULL);
       …
}
void *f1(void *i){
      int a;
      a = * ((int *) i);
}
void *f2(void *i){
      …
}
void *f3() {
      …
}
```

Thread Identification

- To get the ID of the current thread:

- Syntax:

```
pthread_t pthread_self (void)
```

  - Return value:
    - If successful, it returns 0. Otherwise, it returns a nonzero error.

Example

```
pthread_t ntid;
void printds (const char *s)
{
        pid_t pid;
        pthread_t tid;

        pid = getpid();
        tid =pthread_self();
        printf("%s pid %d tid &d\n", s, pid, (unsigned int) tid);
}
void * thr_fn (void * arg)
{
   printds("New Thread: ");
   return(0);
}
int main (void)
{
        int err;
        err = pthread_create(&ntid, NULL, thr_fn, NULL);
        if (err != 0) printf("can't create thread \n");
                printds("main thread: ");
                sleep(1);
        exit(0);
}
```

Thread/Process Termination

- A whole process (with all its threads) terminates if:
  - One of its threads executes an `exit()`
  - One of its threads receives a signal to terminate
  - The main thread executes a `return()`

- A Single thread terminates if:
  - It executes a `pthread_exit()`
  - It executes a `return()` in its initial function
  - It receives a `pthread_cancel()` from another thread

Thread Termination

- To exit a thread:

- Syntax:

```
void pthread_exit (void *retval)
```

- When a thread has finished its work, it can exit by calling the `pthread_exit()` library procedure.

- The `pthread_exit()` function terminates the calling thread and returns a value via `retval` that (if the thread is joinable) is available to another thread in the same process.

- The thread then vanishes and is no longer schedulable and the stack is released.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h> /* POSIX threads */

void *mythread(void *);
        int sharedVar ;      /* global variable */

int main() {
   pthread_t tid;
   sharedVar = 1234;
   printf("Main: sharedVar= %d\n", sharedVar);
   pthread_create(&tid, NULL, mythread, NULL);
   sleep(1); /* yield to another thread */
   printf("Main: sharedVar= %d\n", sharedVar);
   sharedVar = 999;
   printf("Main: sharedVar= %d\n", sharedVar);
   sleep(1); /* yield to another thread */
   printf("Main: sharedVar= %d\n", sharedVar);
   printf("DONE\n");
   exit(0);
}
```

```
void *mythread (void *arg)
{
   /* display the value of the global variable */
   printf("thread %lu: sharedVar is %d \n",
pthread_self(), sharedVar);

   sharedVar = 111;
   printf("thread %lu: sharedVar is %d \n",
pthread_self(), sharedVar);

   sleep(1); /* yield to another thread or main */

   printf("thread %lu: sharedVar is %d \n",
pthread_self(), sharedVar);

   sharedVar = 222;
   printf("thread %lu: sharedVar is %d.\n",
pthread_self(), sharedVar);
   sleep(1); /* yield to another thread or main */

   /* thread exit with an integer */
   pthread_exit (NULL);
}
```

OUTPUT

```
Main: sharedVar= 1234
thread 792: sharedVar is 1234
thread 792: sharedVar is 111
Main: sharedVar= 111
Main: sharedVar= 999
thread 792: sharedVar is 999
thread 792: sharedVar is 222
Main: sharedVar= 222
DONE
```

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define NUM_THREADS 5
void *PrintHello (void *threadid)
{
   long * tid;
   tid = (long *) threadid;
   printf("Hello World! It's me, thread #%ld!\n", * tid);
   pthread_exit(NULL);
}
int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int rc;
long t;
for(t=0;t<NUM_THREADS;t++){
  printf("In main: creating thread %ld\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *) &t);
  if (rc){
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }
pthread_exit(NULL);
}
```

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define NUM_THREADS 5
void *PrintHello (void *threadid)
{
   long * tid;
   tid = (long *) threadid;
   printf("Hello World! It's me, thread #%ld!\n", * tid);
   pthread_exit(NULL);
}
int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int rc;
long t;

for(t=0;t<NUM_THREADS;t++){
  printf("In main: creating thread %ld\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *) &t);
  if (rc){
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }
pthread_exit(NULL);
}
```

OUTPUT

```
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #1!
In main: creating thread 2
Hello World! It's me, thread #2!
Hello World! It's me, thread #3!
In main: creating thread 3
In main: creating thread 4
Hello World! It's me, thread #4!
Hello World! It's me, thread #5!
```

`t` is a pointer and its value changes during the execution

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define NUM_THREADS 5
void *PrintHello (void *threadid)
{
   long * tid;
   tid = (long *) threadid;
   printf("Hello World! It's me, thread #%ld!\n", * tid);
   pthread_exit(NULL);
}
int main(int argc, char *argv[])
{
        pthread_t threads[NUM_THREADS];
        int rc, t;
        long t_numb[NUM_THREADS];

        for(t=0;t<NUM_THREADS;t++){
                t_numb[t] = t;
                printf("In main: creating thread %ld\n", t);
                rc = pthread_create(&threads[t], NULL, PrintHello, (void *) &t_numb[t]);
                if (rc){
                        printf("ERROR; return code from pthread_create() is %d\n", rc);
                        exit(-1);
                }
        }
pthread_exit(NULL);
}
```

OUTPUT

```
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #0!
In main: creating thread 2
Hello World! It's me, thread #1!
Hello World! It's me, thread #2!
In main: creating thread 3
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
```

Multiple Arguments via a structure

- Passing multiple arguments using **structure**:
- Syntax:

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

Passing a structure

```
struct thread_parms {
    int N;
    int start;
    int stop;
};

void* runner(void* param) {
    struct thread_parms* p = (struct thread_parms*) param;
    /* tests all divisors of the assigned interval */
    ...
    ...
    ...
}

int main() {
    struct thread_parms thread_data;
    err = pthread_create(&tid, NULL, &thread Function, (void*)& thread_data);
    ...
    ...
    ...
}
```

Example

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS  5
char *messages[NUM_THREADS];
struct thread_data
{
   int   thread_id;
   int   sum;
   char          *message;
};
struct thread_data thread_data_array[NUM_THREADS];
void *PrintHello(void *threadarg)
{
   int taskid, sum;
   char *hello_msg;
   struct thread_data *my_data;

   sleep(1);
   my_data = (struct thread_data *) threadarg;
   taskid = my_data->thread_id;
   sum = my_data->sum;
   hello_msg = my_data->message;
   printf("Thread %d: %s  Sum=%d\n", taskid, hello_msg, sum);
   pthread_exit(NULL);
}
```

```
int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int *taskids[NUM_THREADS];
int rc, t, sum =0;

messages[0] = "English: Hello World!";
messages[1] = "French: Bonjour, le monde!";
messages[2] = "Spanish: Hola al mundo";
messages[3] = "German: Guten Tag, Welt!";
messages[4] = "Russian: Zdravstvytye, mir!";

for(t=0;t<NUM_THREADS;t++) {
  sum = sum + t;
  thread_data_array[t].thread_id = t;
  thread_data_array[t].sum = sum;
  strcpy(thread_data_array[t].message,messages[t]);
  printf("Creating thread %d\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *)  &thread_data_array[t]);
  if (rc) {
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }
pthread_exit(NULL);
}
```

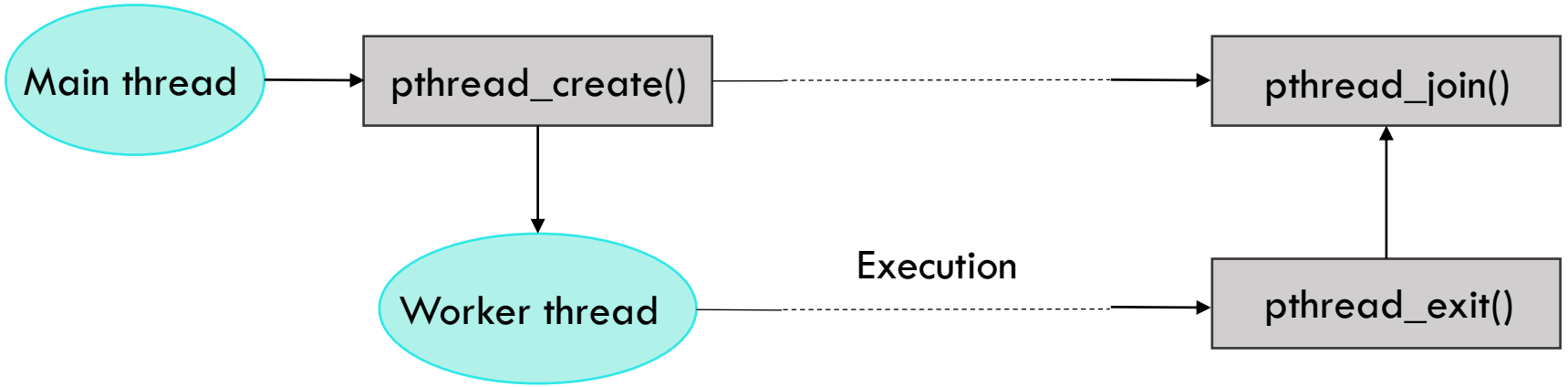
pthread_join()

- At the creation of a thread, it can be declared as:
  - Joinable (default): another thread can execute a "wait" (`pthread_join()`) on it.
  - Detached: it is not possible to wait explicitly its end (it is not joinable), i.e., a detached thread cannot be waited. This attribute is set through the argument `attr` (with the pthread_attr_setdetachstate()) call), as the following code:

```
pthread_attr_t attr;
void *status;
pthread_attr_init (&attr);
pthread_attr_setdetachstate (&attr, PTHREAD_CREATE_DETACHED);
pthread_create(&f1_thread,&attr,f1,&i1);
```

pthread_join()

- In some thread systems, one thread can wait for (a specific) thread to exit by calling the `pthread_join()` procedure.
- This procedure blocks the calling thread until (a specific) thread has exited.



pthread_join()

- To block the calling thread until the specified thread is terminated:

- Syntax:

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

  - `thread` is the id of the thread to wait for
  - `status` is the location where the exit status of the joined thread is stored. This can be NU
    if the exit status is not required.

pthread_join()

- To block the calling thread until the specified thread is terminated:

- Syntax:

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

  - On success, it returns 0. Otherwise, it returns an error code.
  - The result of multiple simultaneous calls to pthread_join() for the same target thread is undefined.
  - If the thread calling pthread_join() is canceled, the target thread is not detached.

pthread_join()

- To block the calling thread until the specified thread is terminated:

- Syntax:

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

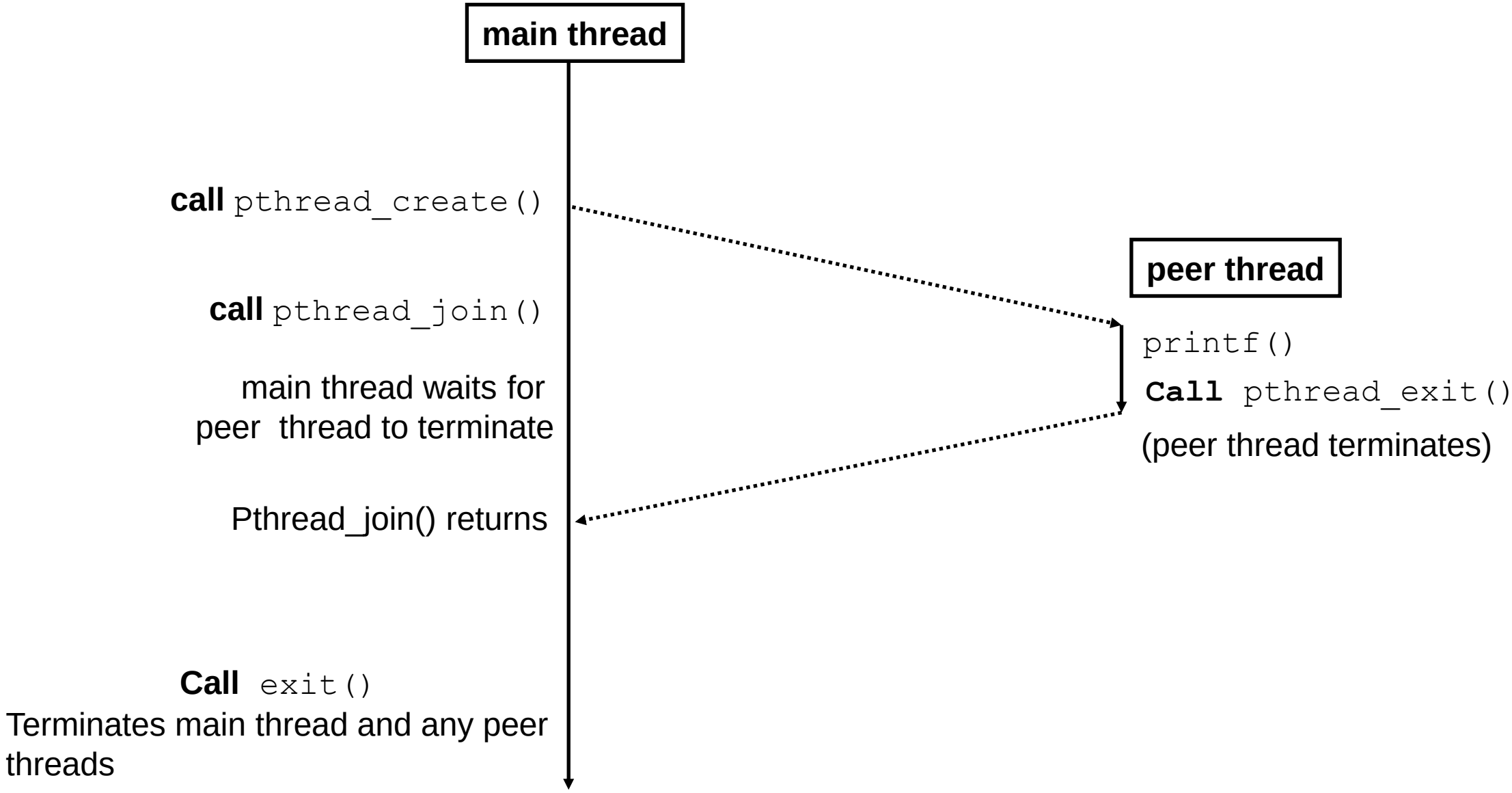
```
void *howdy(void *vargp);

int main() {
  pthread_t tid;

  pthread_create(&tid, NULL, howdy, NULL);
  pthread_join(tid, NULL);
  exit(0);
}

/* thread routine */
void *howdy(void *vargp) {
  printf("Hello, world!\n");
  pthread_exit(NULL);
}
```

Example


main thread
call pthread_create()
call pthread_join()
main thread waits for peer thread to terminate
Pthread_join() returns
Call exit()
Terminates main thread and any peer threads
peer thread
printf()
Call pthread_exit()
(peer thread terminates)

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NTHREADS 5

void *myFun (void *x)
{
  int tid;
  tid = *((int *) x);
  printf ("Hi from thread %d!\n", tid);
  pthread_exit(NULL);
}
```

Output:

```
spawning thread 0
spawning thread 1
Hi from thread 0!
spawning thread 2
Hi from thread 1!
spawning thread 3
Hi from thread 2!
spawning thread 4
Hi from thread 3!
Hi from thread 4!
```

```
int main(int argc, char *argv[])
{
  pthread_t threads[NTHREADS];
  int thread_args[NTHREADS];
  int rc, i;

  /* spawn the threads */
  for (i=0; i<NTHREADS; ++i)
   {
     thread_args[i] = i;
     printf("spawning thread %d\n", i);
     rc = pthread_create(&threads[i], NULL, myFun, (void *) &thread_args[i]);
   }

  /* wait for threads to finish */
  for (i=0; i<NTHREADS; ++i) {
    rc = pthread_join(threads[i], NULL);
  }

  return 1;
}
```

Example

- A program implementing the summation function where the summation operation is run as a separate thread.

```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */
int main(int argc, char *argv[])
{
pthread_t tid; /* the thread identifier */

if (argc != 2) { fprintf(stderr,"usage: a.out <integer value>\n");
        return -1; }

if (atoi(argv[1]) < 0) {
                        fprintf(stderr,"Argument %d must be non-negative\n",atoi(argv[1]));
                        return -1;
                        }
/* create the thread */
pthread_create(&tid,NULL,runner,argv[1]);

/* now wait for the thread to exit */
pthread_join(tid,NULL);

printf("sum = %d\n",sum);
}
```

```
/**
 * The thread will begin control in this function
 */

void *runner(void *param)
{
int i, upper = atoi(param);
sum = 0;

        if (upper > 0) {
                for (i = 1; i <= upper; i++)
                sum += i;
        }

        pthread_exit(NULL);
}
```

Example

- Write a counting program, which should have 2 threads
- While the peer thread loops, incrementing a counter, the main thread peeks at the counter every second and prints its value
- After 10 iterations, the main thread kills the peer one and computes the average number of counts.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <errno.h>

char running = 1;

long long counter = 0;

void * process()
{
 while (running)
    counter++;

printf("Thread: exit\n");

pthread_exit(NULL);
}
```

```
int main()
{
  int i;
  pthread_t thread_Id;
  if (pthread_create(&thread_Id, NULL, process, NULL))
      {
      printf("ERROR in pthread_create()\n");
      exit(-1);
      }

  for (i=0 ; i < 10 ; i++)
      {
      sleep(1);
      printf("counter = %lld\n", counter);
      }

   running = 0;

   pthread_join(thread_Id,  NULL);

   printf("Average Instructions = %lld \n", counter/10);

   return 0;
}
```

Thread and Processes

```
#include <pthread.h>
#include <stdio.h>
#include <sys/types.h>

int value = 0; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

void *runner(void *param)
{
        value = 5;

        pthread_exit(NULL);
}
```

```
int main(int argc, char *argv[])
{
pid_t pid;
pthread_t tid; /* the thread identifier */

pid = fork();
if (pid == 0) { /* child process */
                pthread_create(&tid, NULL,runner,NULL);
                        /* now wait for the thread to exit */
                pthread_join(tid,NULL);
                printf("CHILD: value = %d\n",value);
}
else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d\n",value);
}
}
```

pthread_cancel()

- To terminate a specific thread:

- Syntax:

```
int pthread_cancel (pthread_t *thread_id);
```

  - `Thread_id` is the id of the thread to wait for
  - It returns 0 in case of success and an error code, in case of failure.

pthread_detach()

- Syntax:

```
int pthread_detach (pthread_t *thread_id);
```

  - `thread_id` is the id of the thread to wait for
  - It returns 0 in case of success and an error code, in case of failure.

  - The `pthread_detach()` function marks the thread identified by thread as detached. When a detached thread terminated, its resources are automatically released back to the system without the need to another thread to join with the terminated thread.

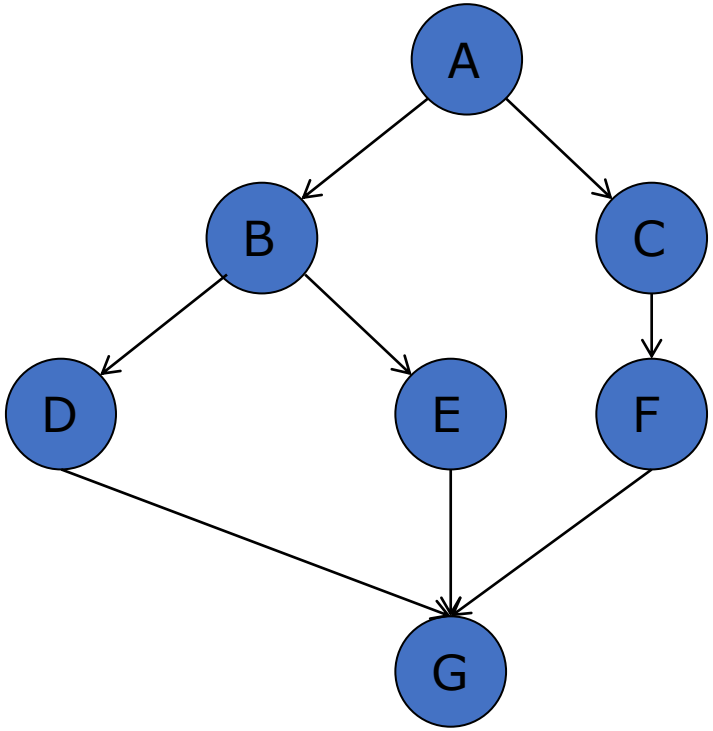
  - A detached thread cannot be joined:
    - All the future calls `pthread_join` to thread `thread_id` will fail with an error code.

Example

```
pthread_t tid;
int rc;
void *status;
rc = pthread_create (&tid, NULL, PrintHello, NULL);
if (rc) { ... }
...
pthread_detach (tid); /* Detach a thread */
rc = pthread_join (tid, &status);
if (rc) {
// Error
exit (-1);
}
pthread_exit (NULL);
```

Example

- Execute a code source able to guarantee the following graph of precedence.



Thread and Processes

```
void waitRandomTime (int max){
sleep ((int)(rand() % max) + 1);
}

int main (void) {
pthread_t th_cf, th_e;

srand (getpid());
waitRandomTime (10);
printf ("A\n");
waitRandomTime (10);
pthread_create (&th_cf,NULL,CF,NULL);
waitRandomTime (10);
printf ("B\n");
waitRandomTime (10);
pthread_create (&th_e,NULL,E,NULL);
waitRandomTime (10);
printf ("D\n");
pthread_join (th_cf, NULL);
pthread_join (th_e, NULL);
waitRandomTime (10);
printf ("G\n“);
return 0;
}
```

Example

- Reading the return value of pthread_exit()

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

pthread_t tid[2];
int ret1,ret2;

void* doSomeThing(void *arg)
{
    unsigned long i = 0;
    pthread_t id = pthread_self();

    sleep(5);
    if(pthread_equal(id,tid[0])) /* The pthread_equal() function compares two thread identifiers.
                                    If the two thread IDs are equal, pthread_equal() returns a nonzero value; otherwise, it returns 0. */
        {
        printf("\n First thread processing done\n");
        ret1  = 100;
        pthread_exit(&ret1);
    }
    else
    {
```

Example

- Reading the return value of pthread_exit()

```
int main(void)
{
    int i = 0;
    int err;
    int *ptr[2];

    while(i < 2)
    {
        err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
        if (err != 0)
            printf("\ncan't create thread :[%s]", strerror(err));
        else
            printf("\n Thread created successfully\n");

        i++;
    }

    pthread_join(tid[0], (void**)&(ptr[0]) );
    pthread_join(tid[1], (void**)&(ptr[1]) );

    printf("\n return value from first thread is [%d]\n", *ptr[0]);
    printf("\n return value from second thread is [%d]\n", *ptr[1]);
     return 0;
}
```

Example

```
static void *CF () {
waitRandomTime (10);
printf ( "C\n");
waitRandomTime (10);
printf ( "F\n");
return ((void *) 1); // Return code
}
static void *E () {
waitRandomTime (10);
printf ("E\n");
return ((void *) 2); // Return code
}
```

Homework

- Execute a code source able to guarantee the following graph of precedences