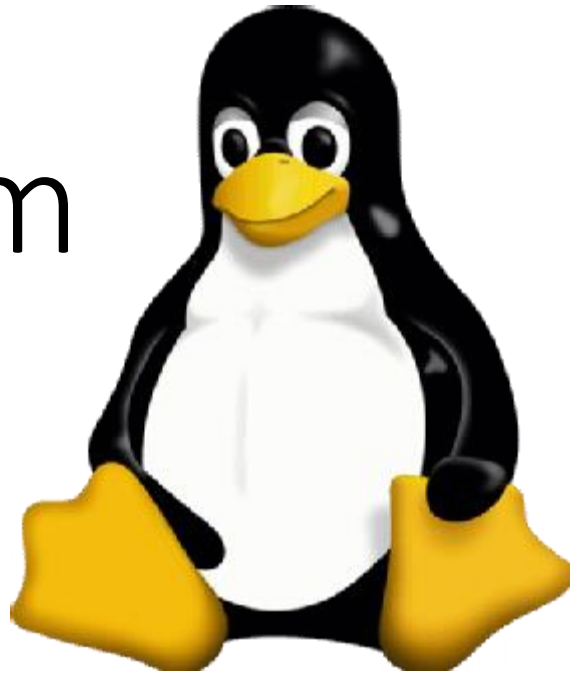
Linux Operating System



Process

- A **process** in Linux is nothing but a program in execution.
- It’s a running instance of a program.
- Any command that you execute starts a process.
- Types of process

1. **Foreground Processes**
   depend on the user for input
   also referred to as interactive processes
2. **Background Processes**
   run independently of the user
   referred to as non-interactive or automatic processes

Process...

- If we want to start a process in the background, then we need to append the command in the Bash shell by &.
- Example : If I want to start my program **Hello** as the background process, then the command would be as follows:

  $ Hello &
- If we terminate any command by &, then it starts running as the background process

Process

- Syntax for background and foreground processes:

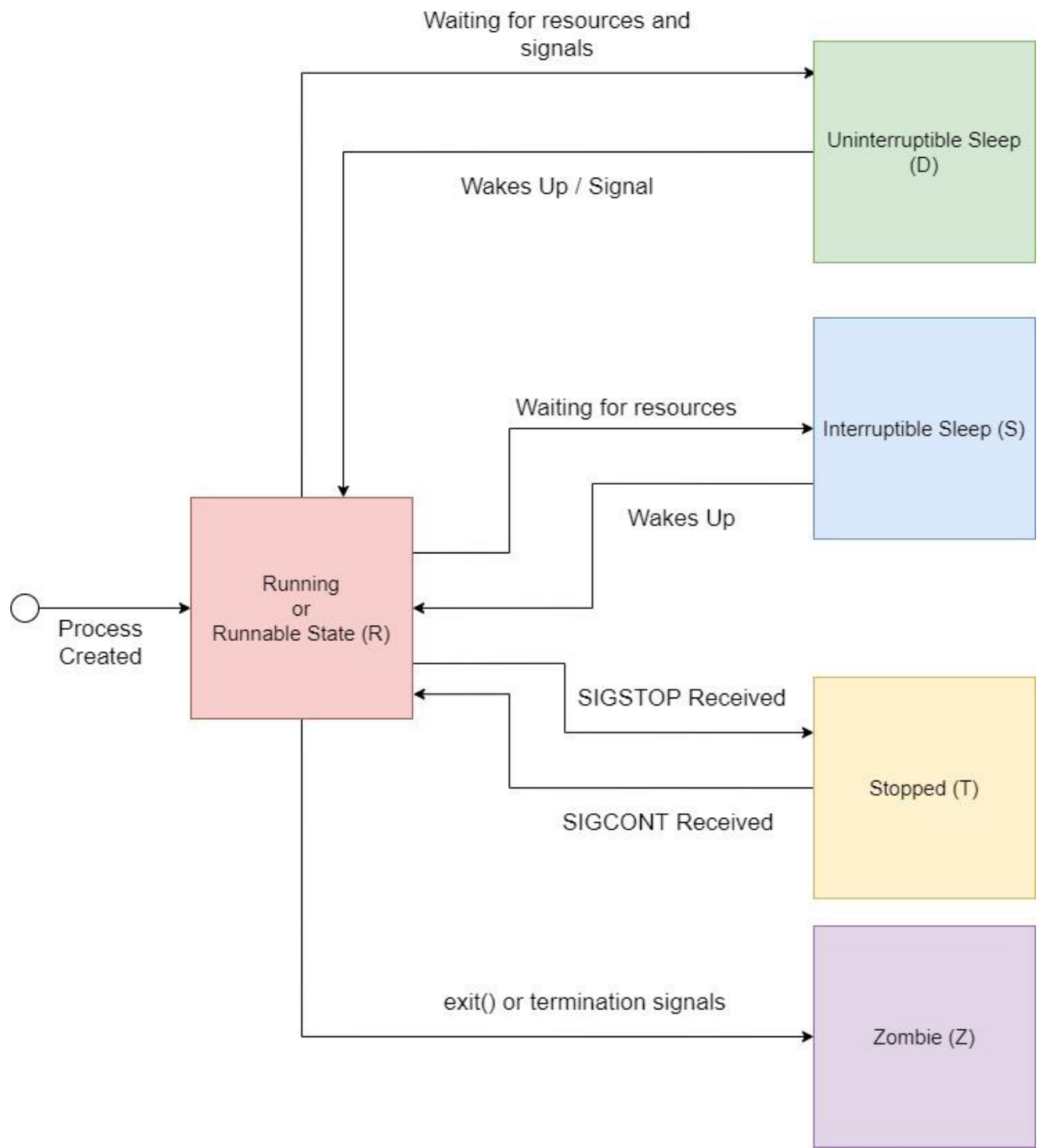
  bg [JOB_SPEC]

  fg [JOB_SPEC]

  Where JOB_SPEC can be any of the following:
  - %n - where *n* is the job number
  - %abc - refers to a job started by a command beginning with *abc*
  - %?abc - refers to a job started by a command containing *abc*
  - %- - specifies the previous job
- Note: *bg* and *fg* operate on the current job if no JOB_SPEC is provided.

Process States

- **Running or Runnable(R)**
- **Sleeping**
  - **Interruptible sleep (S)**
  - **Uninterruptible sleep (D)**
- **Stopped (T)**
- **Zombie (Z)**



Commands for Process Management

- Two commands to track running processes.
- **top**
  - To track the running processes on our machine
  - Top command displays a list of processes that are running in real-time along with their memory and CPU usage.
- **ps**
  - Stands for ‘Process Status’
  - It displays the currently-running processes.
  - However, unlike the top command, the output generated is not in real-time.
  - To list the process associated with our current Bash shell terminal: **$ ps**
  - To check the presence of ‘init’ process : **$ ps –ef**
  - To get more information using ps command use: **$ ps –u** or **man ps**
  - We can also use it to list all the processes: **$ ps –A**

ps Command

- Basic Syntax

  $ ps [options]
- Listing All Processes
  - **$ ps –f**
  - **$ ps –a**
  - **$ ps –x**
  - **$ ps –e**
  - **$ ps –u**
- Simple Filtering
  - **$ ps –C sytemd**
  - **$ ps –p 1, 1658**
  - **$ ps –u root**

ps Command…

- **In Linux, there are two types of user names:**
  - The **real user name** is the one that started the process.
  - The **effective user** is the one that owns the executable behind the process.
- The *ps* command can filter by **real user names with the *-U* option** and by **effective user names with the *-u* option**.

Commands for Process Management

- **Pidof**
  - to track a process by its name
  - displays the PIDs of the processes when it is used with the process name.
- Syntax

  $ pidof process_name

Commands for Process Management

- **Terminate a Process**
  - To stop a process in Linux, use the '**kill'** command.
  - It sends a signal to the process.
  - There are different types of signals that you can send. However, the most common one is 'kill -9' which is '**SIGKILL**'.
  - We can list all the signals using: **$kill –L**
  - **Kill is a native Linux command which sends a signal to specified processes causing them to act according to the signal.**

Terminate a Process…

- The syntax for killing a process is:

  $kill [pid] or

  $kill -9 [pid]

- **Killall**
  - the easiest technique to kill a process if you know the exact process name
  - **$ killall firefox**
- **pkill**
  - If we do not know the exact name of the process
  - **$ pkill fire**

Change priority of a process

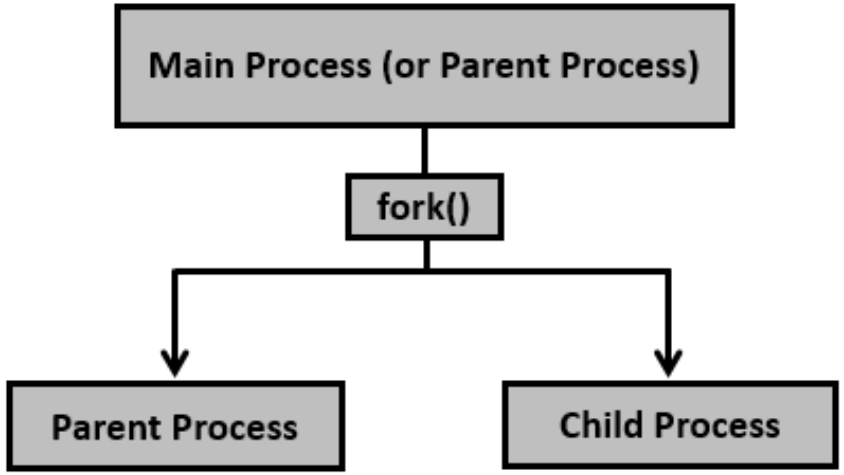
- In Linux, you can prioritize between processes.
- The priority value for a process is called the ‘Niceness’ value.
- Niceness value can range from -**20** to **19**.
- **0** is the default value.
- To start a process and give it a nice value other than the default one, use:

  $nice –n [value] [process name]
- To change nice value of a process that is already running use:

  $renice [value] –p ‘PID’

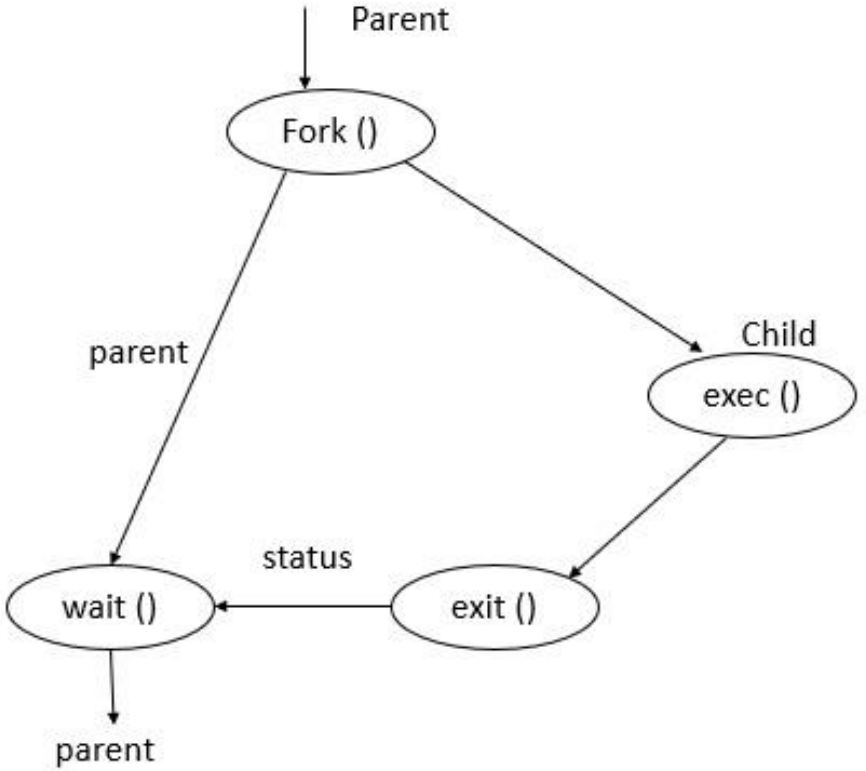
Process Management

- A system call used to create a new process or duplicate process is called a **fork()**.
- The newly created process is called the **child process**
- The process that initiated it is called the **parent process**.



Process Management using System Calls

- Process management system calls in Linux.
  - **fork()** – For creating a duplicate process from the parent process.
  - **wait()** – Processes are supposed to wait for other processes to complete their work.
  - **exec()** – Loads the selected program into the memory.
  - **exit()** – Terminates the process.



To Compile and Run C Program in Linux

1. Check gcc compiler version using: **gcc version--**
2. **touch hello.c** command in the terminal will create an empty hello.c C program file in the desktop directory.
3. Open the ***hello.c*** file in the in-built text editor
4. Type the C Code
5. To compile C Program in Linux, you can use the below command in the terminal: **gcc hello.c –o hello**
6. To run the executable file : **./hello**

Program1

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();
        printf("Called fork() system call\n");
        return 0;
}
```

Program 2

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();
        fork();
        fork();
        printf("hello\n");
        return 0;
}
```

Program 3

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
pid_t pid, mypid, myppid;
pid = getpid();
printf("Before fork: Process id is %d\n", pid);
pid = fork();
if (pid < 0)
{
        printf("fork() failure\n");
        return 1;
 }
# Child process
else if (pid == 0)
{
        printf("This is child process\n");
        mypid =  getpid();
        myppid = getppid();
        printf("Process id is %d and PPID is %d\n",
        mypid, myppid);
}
# Parent process
else
{
        wait(NULL);
        printf("This is parent process\n");
        mypid = getpid(); myppid = getppid();
        printf("Process id is %d and PPID is %d\n",
        mypid, myppid);
        printf("Newly created process id or child pid
        is %d\n", pid);
}
return 0;
}
```

fork() vs exec()

- The fork system call creates a new process.
- The new process created by fork() is a copy of the current process except for the returned value.
- The exec() system call replaces the current process with a new program.

Wait() System Call

- A call to wait() blocks the calling process until one of its child processes exits or a signal is received.
- After child process terminates, parent ***continues*** its execution after wait system call instruction.
- Syntax

```
pid_t wait(int *stat_loc);
```

Program 4

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
        pid_t cpid;
        if (fork()== 0)
                exit(0);                        /* terminate child */
        else
                cpid = wait(NULL);      /* reaping parent */
        printf("Parent pid = %d\n", getpid());
        printf("Child pid = %d\n", cpid);
        return 0;
}
```

Program 4

```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
        if (fork()== 0)
                printf("HC: hello from child\n");
        else
        {
                printf("HP: hello from parent\n");
                wait(NULL);
                printf("CT: child has terminated\n");
        }
        printf("Bye\n");
        return 0;
}
```

exit() System Call

- A process can terminate in either of the two ways:
  - Abnormally, occurs on delivery of certain signals, say terminate signal.
  - Normally, using _exit() system call (or _Exit() system call) or exit() library function.
- The difference between _exit() and exit() is mainly the cleanup activity.
- The **exit()** does some cleanup before returning the control back to the kernel
- The **_exit()** would return the control back to the kernel immediately.

Lab Session 3

1. Implementation of process management using the following system calls of Linux operating system : fork, getpid, exit, wait
2. Write a program using the I/O system calls of Linux operating system (open, read, write, etc)

Lab Session 4

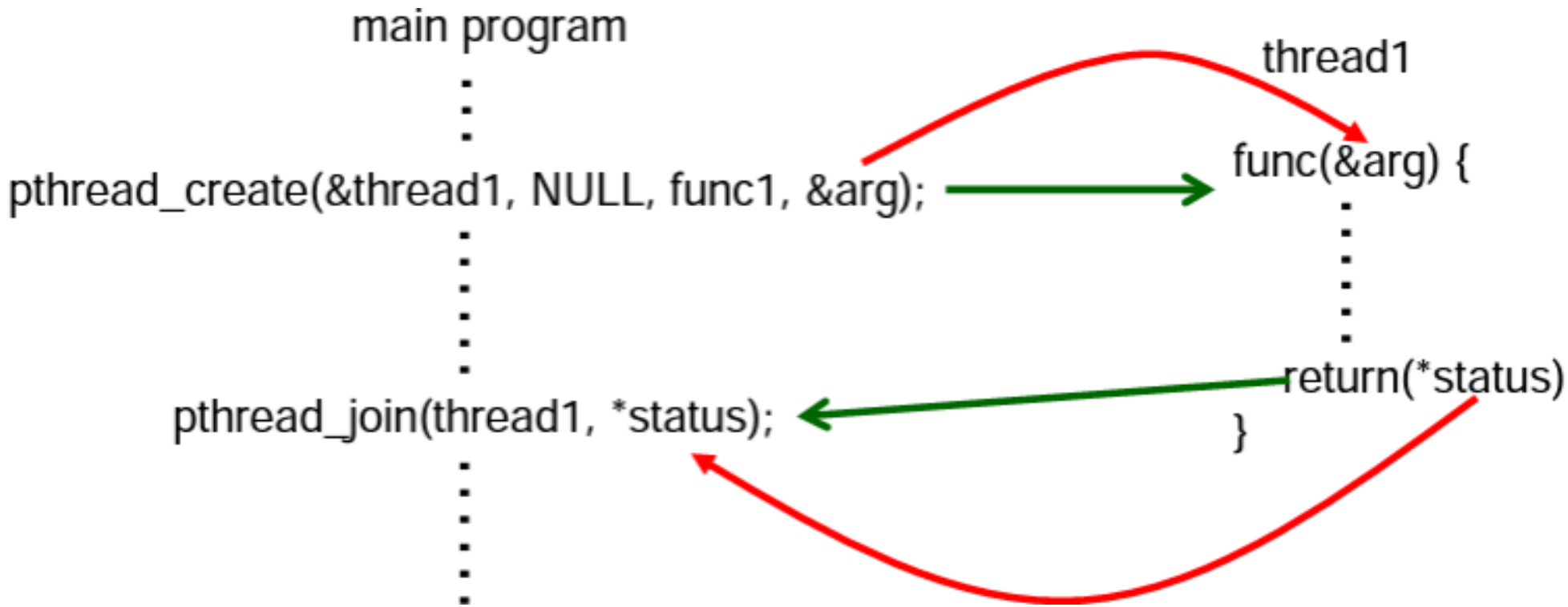
PThreads

- On Linux, you can create and manage threads in C/C++ using the POSIX thread (pthread) library.
- Threads use the same address space and file descriptors as the main program.

1. **Thread Creation**
   - Use the **pthread_create** function to create a new thread.
   - The **pthread.h** header file includes its signature definition
   - **pthread_create** function creates a new thread and makes it executable. It can be called any nnumber of times within the program.
   - Syntax

```
int pthread_create(pthread_t *tid, const pthread_attr_t *attr, void *(*func)(void *), void *arg)
```

     - The first argument is the variable where its thread ID will be stored
     - The second argument contains attributes describing the thread. You can usually just pass a NULL pointer.
     - The third argument is a pointer to the function you want to run as a thread.
     - The final argument is a pointer to data you want to pass to the function.


main program
thread1
pthread_create(&thread1, NULL, func1, &arg);
func(&arg) {
return(*status)
}
pthread_join(thread1, *status);

Pthreads…

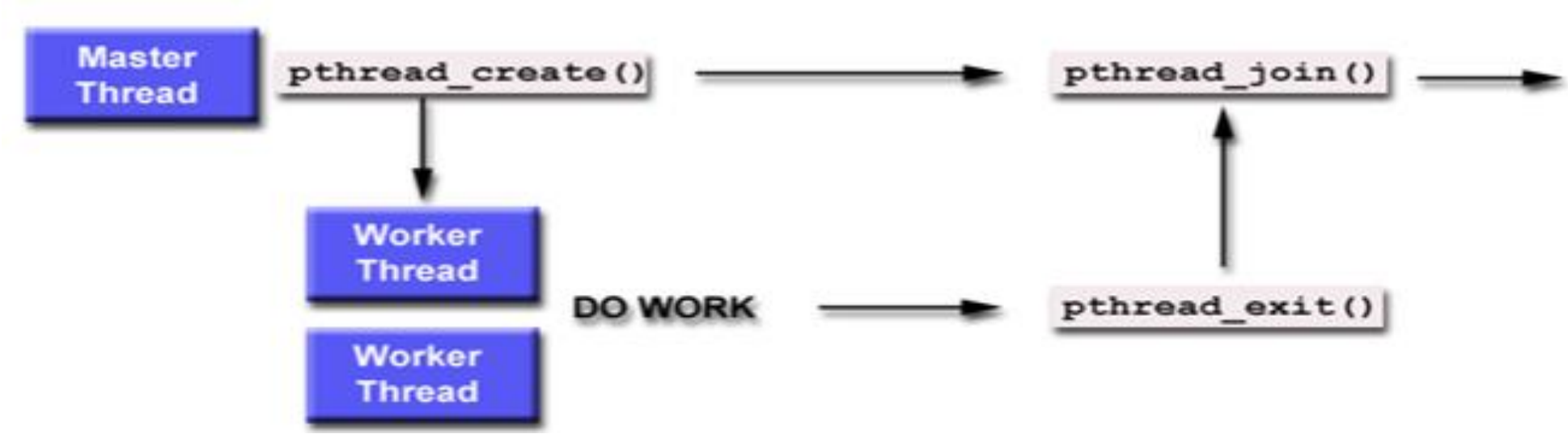
- **Terminating Threads**
  - To exit from a thread, you can use the **pthread_exit** function.
  - Syntax: **void pthread_exit(void *status)**
- **Thread joining**
  - One way to accomplish synchronization between threads
  - Use **pthread_join** function, to wait for a thread to terminate
  - Blocks until the specified thread Id thread terminates
  - Syntax: **int pthread_join(pthread_t tid, void **status)**
    - The first argument is the thread ID.
    - The second argument is a pointer to the data your thread function returned.
  - Example: to create a pthread barrier
    - **for (int i=0; i<n; i++) pthread_join(thread[i], NULL);**

Pthreads…

- **Thread Detaching**
  - Once a thread is detached, it can never be joined
  - Detach a thread could free some system resources
  - Syntax: int pthread_detach(pthread_t thread);

- The thread function can use the **pthread_self** function to return its thread ID.

Multithreaded Program



Example 1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>.
#include <pthread.h>
void *myThreadFun(void *vargp)
{
        sleep(1);
        printf("Hello from Thread\n");
        return NULL;
}
```

```
int main()
{
        pthread_t thread_id;
        printf("Before Thread\n");
        pthread_create(&thread_id,
NULL, myThreadFun, NULL);
        pthread_join(thread_id,
NULL);
        printf("After Thread\n");
        exit(0);
}
```

Compiling Thread Program…

- To compile a multithreaded program using gcc, we need to link it with the pthreads library.
- You can instruct the compiler to link to the library using the **-l** option as:

```
gcc multithread.c -lpthread
```

Pthreads Code for Joining 10 Threads

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

Example 1(version1)

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
void *PrintHello(void *threadId)
{
        long* data = static_cast <long*>
        threadId;
        printf("Hello World! It's me,
        thread #%ld!\n", *data);
        pthread_exit(NULL);
}
```

```
int main (int argc, char *argv[])
{
        pthread_t
        threads[NUM_THREADS];
        for(long tid=0;
tid<NUM_THREADS; tid++)
{
        pthread_create(&threads[tid],
NULL, PrintHello, (void *)&tid);
}   /* Last thing that main() should do
*/
pthread_exit(NULL);
}
```

Example 2

```
#include<pthread.h>
#include<stdio.h>
int sum;
/* the thread function*/
void*runner(void*param);
int main (intargc, char*argv[])
{
        pthread_tt id;
        pthread_attr_t attr;
```

```
/* Get the default attributes */
pthread_attr_init(&attr);
/* create the thread*/
pthread_create(&tid,&attr,runner,
argv[1]);
/* nowwaitforthethreadtoexit*/
pthread_join(tid,NULL);
printf("sum= %d\n",sum);
}
```

```
/* Thethreadfunction*/
void*runner(void*param)
{
        int i,
        upper= atoi(param);
        sum= 0;
        if(upper> 0)
        {
        for(i = 1; i <= upper; i++)
                sum+= i;
        }
        pthread_exit(0);
}
```

Example 3

```
#include<pthread.h>
#include<stdio.h>
int sum;
void*runner(void*param);
#define NUM_THREADS 10
Int main (intargc, char*argv[]) {
        int i;
        pthread_tworkers[NUM_THREADS];        /* thethreadarray*/
        pthread_attr_t attr;
        sum= 0;
        pthread_attr_init(&attr);
```

```
/* createthethread*/
for(i=0; i<NUM_THREADS; i++)
        pthread_create(&worker[i],
        &attr,runner, i+1);
/* nowwaitforthethreadtoexit*/
for(i=0; i<NUM_THREADS; i++)
        pthread_join(worker[i]
,NULL);
printf("sum= %d\n",sum); }
```

Example 3…

```
/* The thread function*/
void*runner(void*param)
{
        int i, upper= atoi(param);
        if(upper> 0)
        {
        for(i = 1; i <= upper; i++)
                sum+= i;
        }
        pthread_exit(0);
}
```

Exercise

- Write a multithreaded program to compute and display the sum of first N natural numbers, the sum of first N odd natural numbers and the sum of first N even natural numbers