# Operating Systems

***CSN-232***

**Dr. R. Balasubramanian**

**Associate Professor**

**Department of Computer Science and Engineering**

**Indian Institute of Technology Roorkee**

**Roorkee 247 667**

*balarfcs@iitr.ac.in*

*https://sites.google.com/site/balaiitr/*

# Process Termination

- Some operating systems do not allow child to exists if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the **`wait()`** system call**.** The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

- If no parent waiting (did not invoke **`wait()`**) process is a **zombie**
- If parent terminated without invoking **`wait`** , process is an **orphan**

# Zombie Process

```c
// A C program to demonstrate Zombie Process.
// Child becomes Zombie as parent is sleeping
// when child process exits.
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    // Fork returns process id
    // in parent process
    pid_t child_pid = fork();

    // Parent process
    if (child_pid > 0)
        {printf("in parent process");
        sleep(50);
        }

    // Child process
    else
        {   printf("in child process");
            exit(0);
        }

    return 0;
}
```

```
$gcc -o main *.c
$main
in child process
```

**http://tpcg.io/6ZccnX**

codingground
SIMPLY EASY CODING

zombie1 (GNU GCC v7.1.1)

Execute | Embed | main.c | Stdin

```c
// A C program to demonstrate working of
// fork() and process table entries.
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>

int main()
{
    int i;
    int pid = fork();

    if (pid == 0)
    {
        for (i=0; i<20; i++)
            printf("I am Child\n");
    }
    else
    {
        printf("I am Parent\n");
        while(1);
    }
}
```

**http://tpcg.io/6ZccnX**

```
$gcc -o main *.c
$main
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
```

```c
// A C program to demonstrate working of
// fork() and process table entries.
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>

int main()
{
    int i;
    int pid = fork();

    if (pid == 0)
    {
        for (i=0; i<20; i++)
            printf("I am Child\n");
    }
    else
    {
        printf("I am Parent\n");
        while(1);
    }
}
```

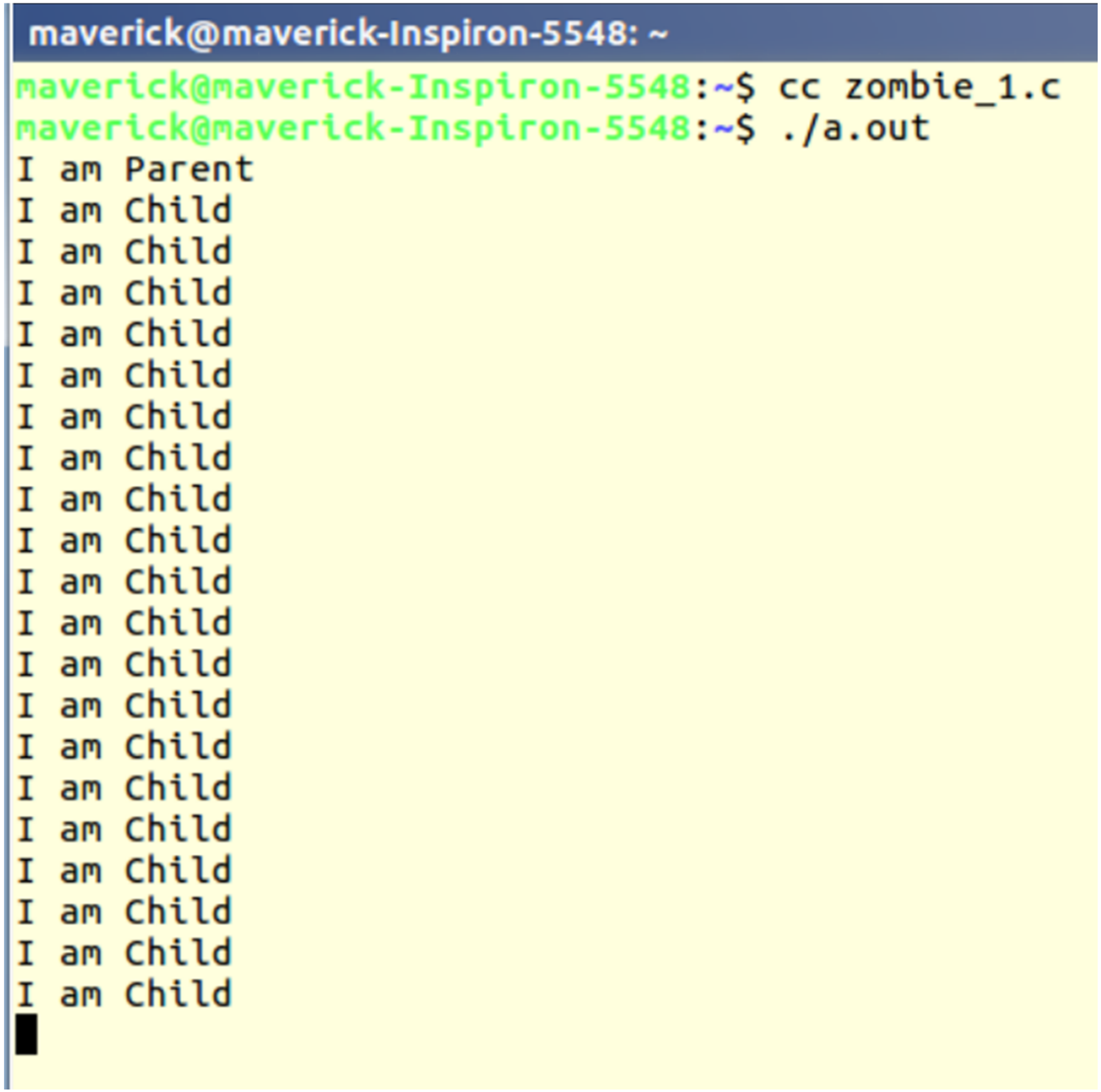
Default Term | + | Browser

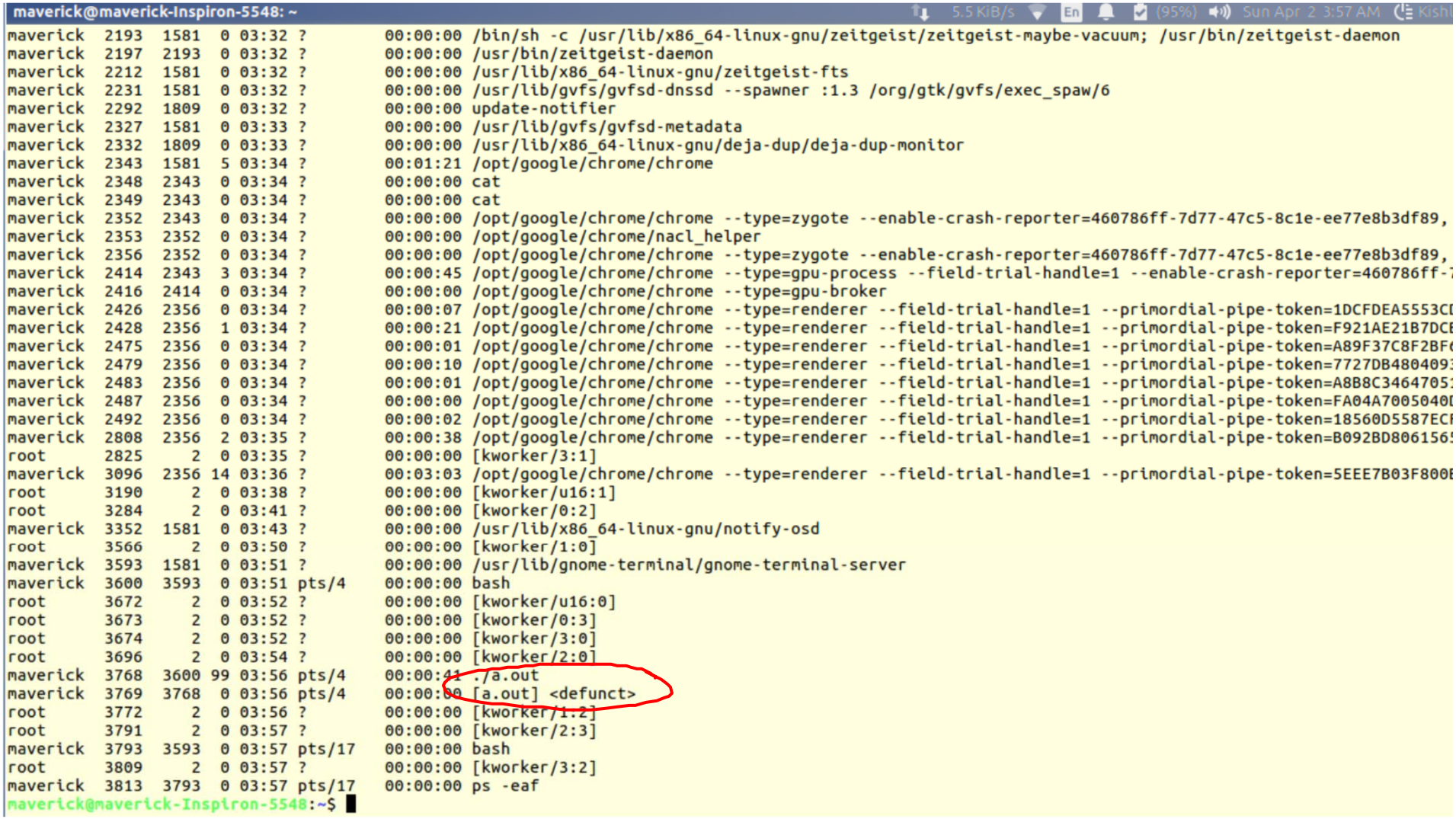
```
sh-4.4$ gcc -o main *.c
sh-4.4$ main
I am Parent
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
```

**https://goo.gl/TwJJWP**

# [main] defunct

```
[1]+  Stopped(SIGTSTP)         main
sh-4.4$ ps -eaf
UID          PID  PPID  C STIME TTY          TIME CMD
6989           1     0  0 17:08 ?        00:00:01 --CODINGGROUND--
6989          13     1  0 17:08 pts/0    00:00:00 sh
6989          19     1  0 17:08 pts/1    00:00:00 sh
6989          26    19 66 17:08 pts/1    00:05:55 main
6989          27    26  0 17:08 pts/1    00:00:00 [main] <defunct>
6989          28    19  0 17:17 pts/1    00:00:00 ps -eaf
sh-4.4$
```


maverick@maverick-Inspiron-5548: ~
maverick@maverick-Inspiron-5548:~$ cc zombie_1.c
maverick@maverick-Inspiron-5548:~$ ./a.out
I am Parent
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child


maverick@maverick-Inspiron-5548: ~
5.5 KiB/s
En
(95%)
Sun Apr 2 3:57 AM
maverick 2193 1581 0 03:32 ? 00:00:00 /bin/sh -c /usr/lib/x86_64-linux-gnu/zeitgeist/zeitgeist-maybe-vacuum; /usr/bin/zeitgeist-daemon
maverick 2197 2193 0 03:32 ? 00:00:00 /usr/bin/zeitgeist-daemon
maverick 2212 1581 0 03:32 ? 00:00:00 /usr/lib/x86_64-linux-gnu/zeitgeist-fts
maverick 2231 1581 0 03:32 ? 00:00:00 /usr/lib/gvfs/gvfsd-dnssd --spawner :1.3 /org/gtk/gvfs/exec_spaw/6
maverick 2292 1809 0 03:32 ? 00:00:00 update-notifier
maverick 2327 1581 0 03:33 ? 00:00:00 /usr/lib/gvfs/gvfsd-metadata
maverick 2332 1809 0 03:33 ? 00:00:00 /usr/lib/x86_64-linux-gnu/deja-dup/deja-dup-monitor
maverick 2343 1581 5 03:34 ? 00:01:21 /opt/google/chrome/chrome
maverick 2348 2343 0 03:34 ? 00:00:00 cat
maverick 2349 2343 0 03:34 ? 00:00:00 cat
maverick 2352 2343 0 03:34 ? 00:00:00 /opt/google/chrome/chrome --type=zygote --enable-crash-reporter=460786ff-7d77-47c5-8c1e-ee77e8b3df89,
maverick 2353 2352 0 03:34 ? 00:00:00 /opt/google/chrome/nacl_helper
maverick 2356 2352 0 03:34 ? 00:00:00 /opt/google/chrome/chrome --type=zygote --enable-crash-reporter=460786ff-7d77-47c5-8c1e-ee77e8b3df89,
maverick 2414 2343 3 03:34 ? 00:00:45 /opt/google/chrome/chrome --type=gpu-process --field-trial-handle=1 --enable-crash-reporter=460786ff-
maverick 2416 2414 0 03:34 ? 00:00:00 /opt/google/chrome/chrome --type=gpu-broker
maverick 2426 2356 0 03:34 ? 00:00:07 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1 --primordial-pipe-token=1DCFDEA5553C
maverick 2428 2356 1 03:34 ? 00:00:21 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1 --primordial-pipe-token=F921AE21B7DC
maverick 2475 2356 0 03:34 ? 00:00:01 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1 --primordial-pipe-token=A89F37C8F2BF
maverick 2479 2356 0 03:34 ? 00:00:10 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1 --primordial-pipe-token=7727DB480409
maverick 2483 2356 0 03:34 ? 00:00:01 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1 --primordial-pipe-token=A8B8C3464705
maverick 2487 2356 0 03:34 ? 00:00:00 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1 --primordial-pipe-token=FA04A7005040
maverick 2492 2356 0 03:34 ? 00:00:02 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1 --primordial-pipe-token=18560D5587EC
maverick 2808 2356 2 03:35 ? 00:00:38 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1 --primordial-pipe-token=B092BD806156
root 2825 2 0 03:35 ? 00:00:00 [kworker/3:1]
maverick 3096 2356 14 03:36 ? 00:03:03 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1 --primordial-pipe-token=5EEE7B03F800
root 3190 2 0 03:38 ? 00:00:00 [kworker/u16:1]
root 3284 2 0 03:41 ? 00:00:00 [kworker/0:2]
maverick 3352 1581 0 03:43 ? 00:00:00 /usr/lib/x86_64-linux-gnu/notify-osd
root 3566 2 0 03:50 ? 00:00:00 [kworker/1:0]
maverick 3593 1581 0 03:51 ? 00:00:00 /usr/lib/gnome-terminal/gnome-terminal-server
maverick 3600 3593 0 03:51 pts/4 00:00:00 bash
root 3672 2 0 03:52 ? 00:00:00 [kworker/u16:0]
root 3673 2 0 03:52 ? 00:00:00 [kworker/0:3]
root 3674 2 0 03:52 ? 00:00:00 [kworker/3:0]
root 3696 2 0 03:54 ? 00:00:00 [kworker/2:0]
maverick 3768 3600 99 03:56 pts/4 00:00:41 ./a.out
maverick 3769 3768 0 03:56 pts/4 00:00:00 [a.out] <defunct>
root 3772 2 0 03:56 ? 00:00:00 [kworker/1:2]
root 3791 2 0 03:57 ? 00:00:00 [kworker/2:3]
maverick 3793 3593 0 03:57 pts/17 00:00:00 bash
root 3809 2 0 03:57 ? 00:00:00 [kworker/3:2]
maverick 3813 3793 0 03:57 pts/17 00:00:00 ps -eaf
maverick@maverick-Inspiron-5548:~$

# Orphan Process

- A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

```c
// A C program to demonstrate Orphan Process.
// Parent process finishes execution while the
// child process is running. The child process
// becomes orphan.
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    // Create a child process
    int pid = fork();

    if (pid > 0)
        printf("in parent process\n");

    // Note that pid is 0 in child process
    // and negative if fork() fails
    else if (pid == 0)
    {
        sleep(30);
        printf("in child process");
    }

    return 0;
}
```

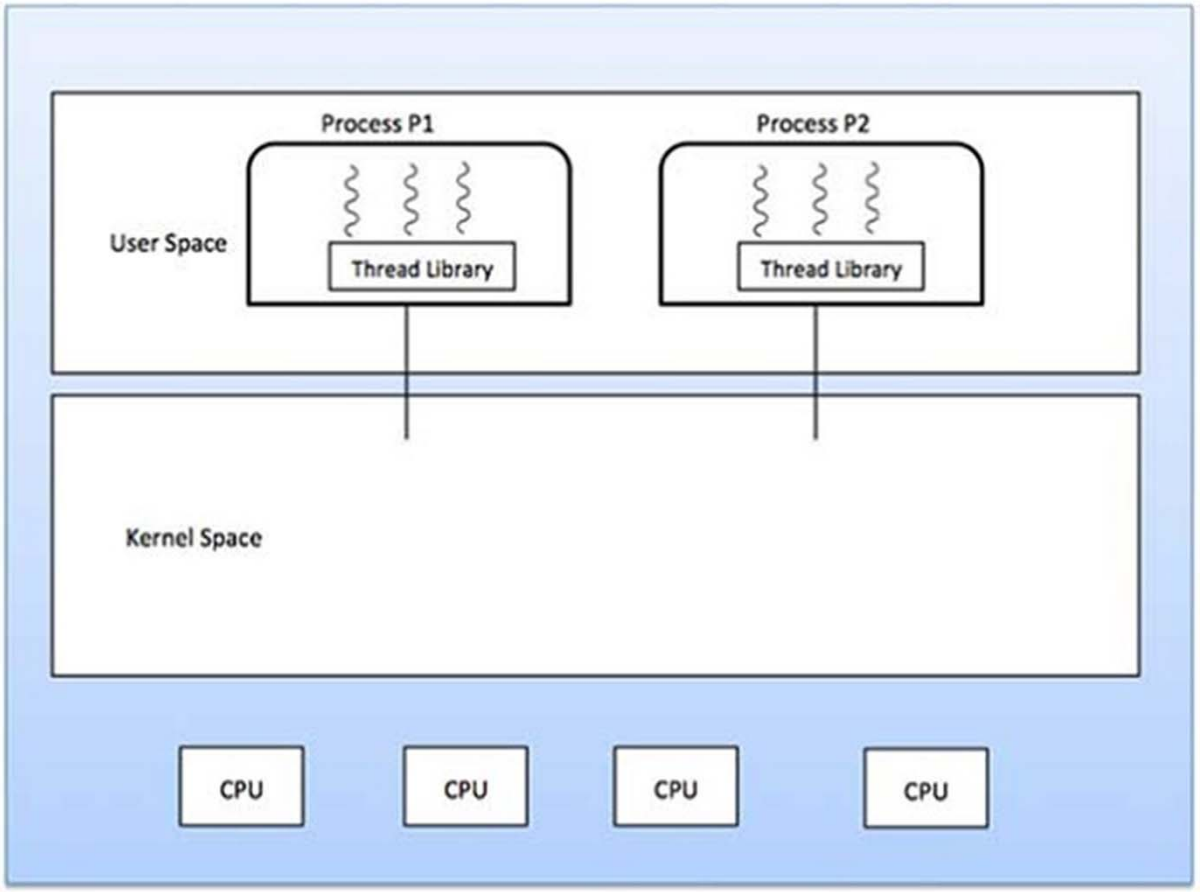
```
$gcc -o main *.c
$main
in parent process
in child process
```

**http://tpcg.io/mAdbKo**

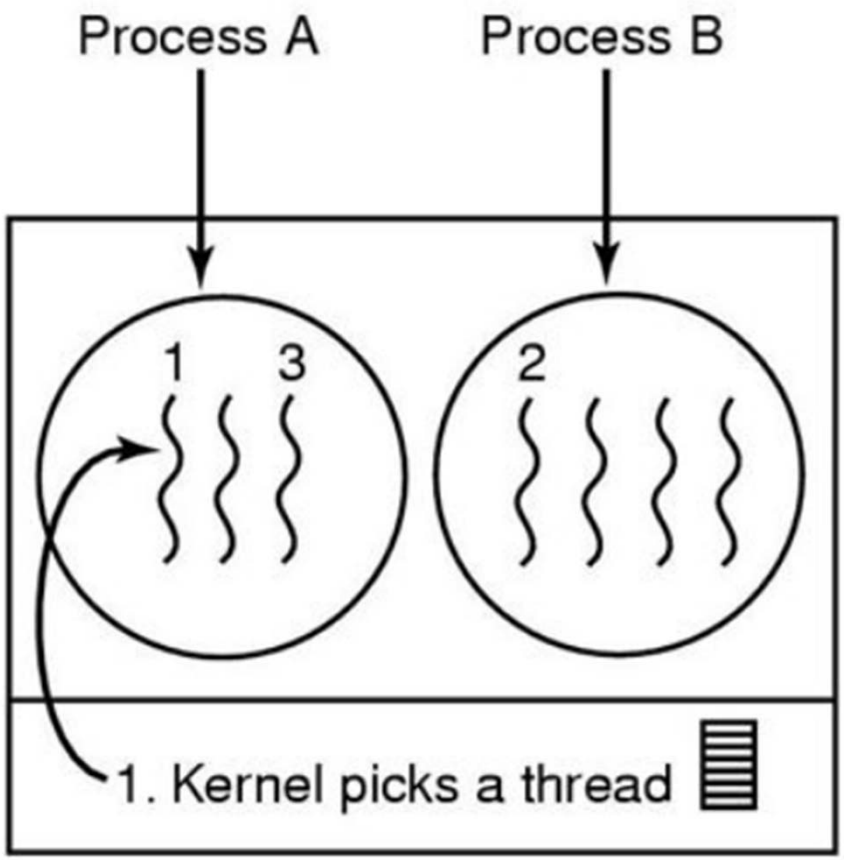
# Threads

- Concurrency
  - Implementing an application as a set of concurrent processes
- Thread
  - Other alternative to processes for concurrent applications
  - Defined as a lightweight process, is a basic unit of CPU utilization, and consists of a program counter, a register set, and a stack space.
  - A process may consist of many threads

# User level Thread (ULT)



# Kernel-level thread scheduling



| USER LEVEL THREAD | KERNEL LEVEL THREAD |
|---|---|
| User thread are implemented by users. | kernel threads are implemented by OS. |
| OS doesn't recognize user level threads. | Kernel threads are recognized by OS. |
| Implementation of User threads is easy. | Implementation of Kernel thread is complicated. |
| Context switch time is less. | Context switch time is more. |
| Context switch requires no hardware support. | Hardware support is needed. |
| If one user level thread perform blocking operation then entire process will be blocked. | If one kernel thread perform blocking operation then another thread can continue execution. |
| Example : Java thread, POSIX threads. | Example : Window Solaris. |

# Commonly used methods of Thread class in JAVA


- **public void run():** is used to perform action for a thread.
- **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
- **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
- **public void join():** waits for a thread to die.
- **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
- **public int getPriority():** returns the priority of the thread.
- **public int setPriority(int priority):** changes the priority of the thread.
- **public String getName():** returns the name of the thread.
- **public void setName(String name):** changes the name of the thread.
- **public Thread currentThread():** returns the reference of currently executing thread.

# More on POSIX

http://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html

# GATE 2017 question

Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and priority (0 is the highest priority) shown below. None of the processes have I/O burst time.

| Process | Arrival Time | Burst Time | Priority |
|---|---|---|---|
| $P_1$ | 0 | 11 | 2 |
| $P_2$ | 5 | 28 | 0 |
| $P_3$ | 12 | 2 | 3 |
| $P_4$ | 2 | 10 | 1 |
| $P_5$ | 9 | 16 | 4 |

The average waiting time (in milliseconds) of all the processes using preemptive priority scheduling algorithm is _____.