**Multithreaded Programming Basics Assignments**

1. **Which applications can be multithreaded?**

**Ans :**The application that to be created for multitasking and multiprocessing in parallel at the same time to reduce the operation or execution time those all application can be multithreaded .

Some examples of multithreaded application:

1. visual basic
2. word application
3. Excel application etc.

In the multithreading process, **each thread runs parallel to each other**. Threads do not allow you to separate the memory area. Therefore it saves memory and offers a better application performance.

1. **Which applications cannot be multithreaded?**

**Ans:** the applications in that it is not possible to share global variable. As each isolate has its own memory,space and everything they can’t be multithreaded.

For example: **LAME,** the **open source audio encoder**, is a good example of a single-threaded application. It will never use more than one thread (so when I'm encoding MP3 files, I just run four or more copies at the same time, each encoding a list of audio files)

1. **.How to create or use POSIX threads?**

Ans: POSIX threads is also known as pthread. Pthread can be created by following the below steps.

POSIX provides **pthread\_create() API** to create a thread

**pthread\_create()** accepts 4 arguments i.e.

1. Pointer of the Thread ID, it will update the value in it.
2. Attributes to set the properties of thread.
3. Function pointer to the function that thread will run in parallel on start. This function should accept a void \* and return void \* too.
4. Arguments to be passed to function.

So, now let’s call pthread\_create() by passing function pointer and other arguments i.e.

**// Thread id**

**pthread\_t threadId;**

**// Create a thread that will function threadFunc()**

**int err = pthread\_create(&threadId, NULL, &threadFunc, NULL);**

**Check if thread is created sucessful**

**pthread\_create()** returns the error code to specify the result of thread creation request. If thread is created successfully then it will return 0. Where as, if thread creation is failed it will return error code to specify the error. We can use **strerror()** to get the detail of error.

**if (err)**

**std::cout << "Thread creation failed : " << strerror(err);**

**else**

**std::cout << "Thread Created with ID : " << threadId << std::endl;**

Main function and other created threads runs in parallel. But when main function ends, complete process exits and all the other thread will also be terminated. Therefore, in main function before ending we should wait for other threads to exit.

1. POSIX Library provides a function for it i.e. **pthread\_join()** to wait for other thread to exit.

**// Wait for thread to exit**

**err = pthread\_join(threadId, NULL);**

**// check if joining is sucessful**

**if (err)**

**std::cout << "Failed to join Thread : " << strerror(err) << std::endl;**

It accepts a thread ID and pointer to store the return value from thread.

1. How to compile and execute program using POSIX threads API. Mention the statements to compile a file mythread.c and generate an executable “mythread”.

**Ans:**

// thread0.c

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

void \*worker\_thread(void \*arg)

{

printf("This is worker\_thread()\n");

pthread\_exit(NULL);

}

int main()

{

pthread\_t my\_thread;

int ret;

printf("In main: creating thread\n");

ret = pthread\_create(&my\_thread, NULL, &worker\_thread, NULL);

if(ret != 0) {

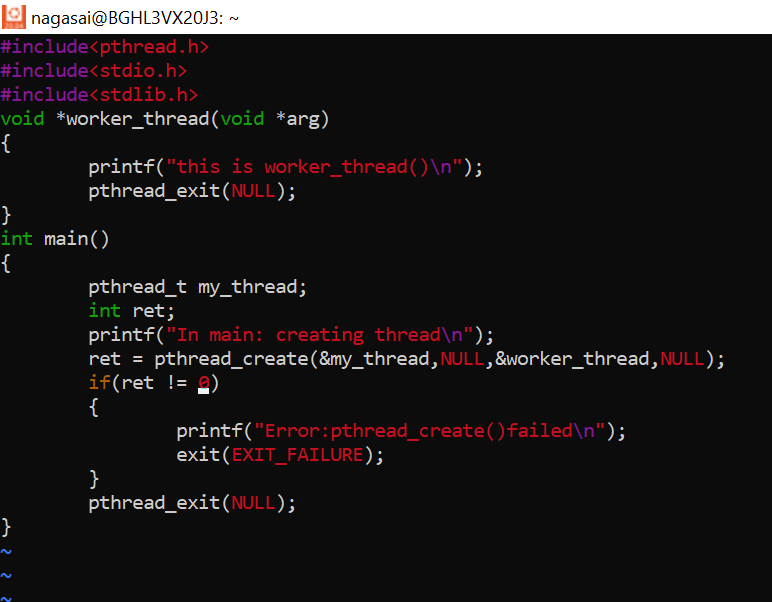
printf("Error: pthread\_create() failed\n");

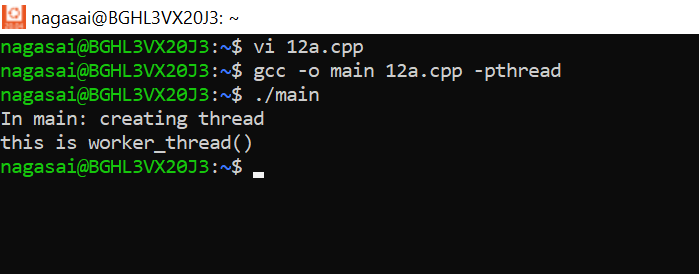
exit(EXIT\_FAILURE);

}

pthread\_exit(NULL);

}





1. **Is mutex same as semaphore?**

**Ans**: signal (mutex); **A Mutex is different than a semaphore as it is a locking mechanism while a semaphore is a signalling mechanism**. A binary semaphore can be used as a Mutex but a Mutex can never be used as a semaphore.

1. **What is the difference between mutex and semaphore? Can we use mutex as a semaphore and vice versa?**

**Ans**: **Semaphore is an integer variable.** **Mutex allows multiple program threads to access a single resource but not simultaneously**. Semaphore allows multiple program threads to access a finite instance of resources. Mutex object lock is released only by the process that has acquired the lock on the mutex object.

No we can’tuse mutex as a semaphore but A binary semaphore can be used as a Mutex but a Mutex can never be used as a semaphore

1. What are the 2 thread standards? Which one is commonly used?
2. What do you mean by starvation and deadlock? When will it occur? What are the solutions to handle that?

**Ans:**

**Starvation :**

Starvation is **the problem that occurs when low priority processes get jammed for an unspecified time as the high priority processes keep executing**. A steady stream of higher-priority methods will stop a low-priority process from ever obtaining the processor.

**Deadlock**

A deadlock is **a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function**. The earliest computer operating systems ran only one program at a time.

**Occurrence:** Starvation occurs when one or more threads in your program are blocked from gaining access to a resource and, as a result, cannot make progress.

Deadlock, the ultimate form of starvation, occurs when two or more threads are waiting on a condition that cannot be satisfied.

**Solution of starvation and deadlock:**

**Random selection of processes for resource allocation or processor allocation should be avoided as they encourage starvation**. The priority scheme of resource allocation should include concepts such as aging, where the priority of a process is increased the longer it waits

Deadlock can be prevented by eliminating any of the four necessary conditions, which are **mutual exclusion, hold and wait, no preemption, and circular wait**. Mutual exclusion, hold and wait and no preemption cannot be violated practically. Circular wait can be feasibly eliminated by assigning a priority to each resource

1. **What all are shared across threads?**

**Ans: The items that are shared among threads within a process are:**

* Text segment (instructions)
* Data segment (static and global data)
* BSS segment (uninitialized data)
* Open file descriptors.
* Signals.
* Current working directory.
* User and group IDs.

1. **What is the difference between thread, process and a program?**

**Ans: A thread is the minimum execution unit of code managed by the operating system**.

**Program**:A process refers to the code and data in memory segments into which the operating system loads a program.

Simply put, **a process is a program that is loaded into the memory to be executed by the processor.**

1. **Is thread id same as PID? How will you get thread id?**

**And**: yes, **Thread and process have the same PID**. Whenever a process spawns a thread or multiple threads, all of them (including process) have the same PID. The difference will be in their TGID (Thread group ID). Threads are identifying by their unique ID called TGID.

To get the thread ID, we call gettid() in Linux.

1. **Name atleast 5 thread attributes**.

**Ans:**

* 1. Priority.
  2. Stack size
  3. Name.
  4. Thread group.
  5. Detach state
  6. Scheduling policy.
  7. Inherit scheduling.

1. Refer link below and answer the question.

<https://man7.org/linux/man-pages/man3/pthread_attr_setscope.3.html>

what are the 2 thread scopes, if there are 4 processes, each with 3 threads then comment on %allocation of CPU to every thread in the 2 thread scope cases.

**Ans:**

1) process local scheduling (known as Process Contention Scope, or Unbound Threads—the Many-to-Many model) and

2) system global scheduling (known as System Contention Scope, or Bound Threads—the One-to-One model).

if there are 4 processes, each with 3 threads then comment on %allocation of CPU to every thread in the 2 thread scope cases.

% of allocation of CPU to every thread in the 2 thread scope cases will be 4.1% for each thread on the basis of pcs and scs scopes.

1. **Which one is memory intensive multithreading or multiprocessing?**

Ans: Multithreading is memory intensive.

1. **Why are threads referred as LWP?**

Ans: Threads are sometimes called lightweight processes because **they have their own stack but can access shared data**. Because threads share the same address space as the process and other threads within the process, the operational cost of communication between the threads is low, which is an advantage.

1. **Can threads of different process communicate with each other? If yes, what are the mechanisms available?**

Ans: No, threads of different process can not communicate with each other.

1. **What is the difference between concurrent and parallel processing?**

**Ans:** Concurrency is the task of running and managing the multiple computations at the same time. While parallelism is the task of running multiple computations simultaneously.