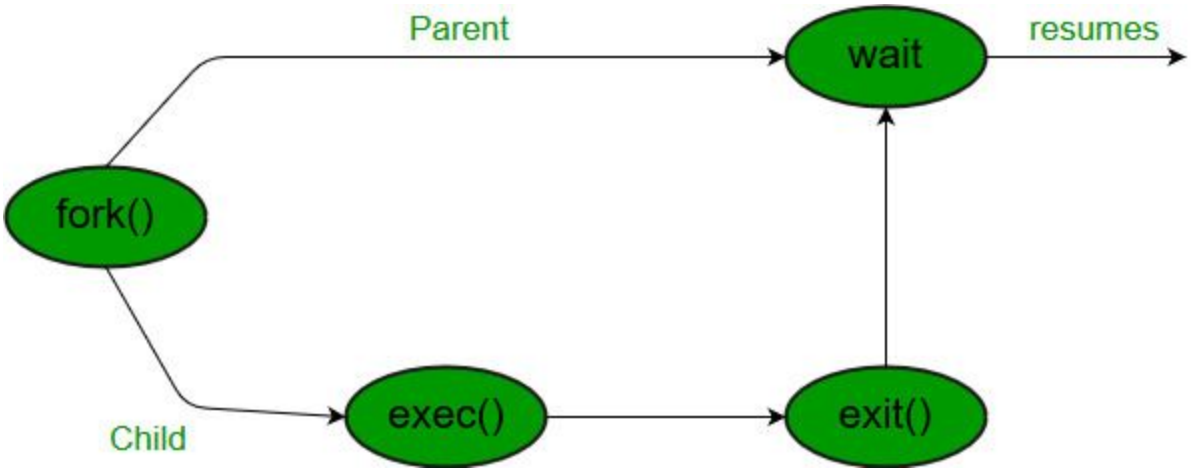
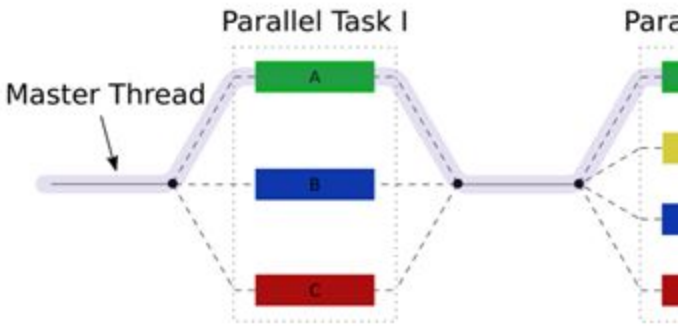
OS Assignment 2 Report

Creation of thread and process to calculate mean, median, mode





Happy Kumar

25.11.2018

CS17BTECH11018 OS1 Assignment 2

Dr. sathya peri

Overview

In program we have done multitasking by using multithreading and multiprocessing. Assgn2-ProcStat-CS17BTECH11018 Program first creates child processes of parent process to calculate average, median and standard deviation. Assgn2-ThStat-CS17BTECH11018 created multithread of a process to calculate average, median, standard deviation.

Implementation

In Assgn2-ProcStat-CS17BTECH11018 program child process is created by using the fork function contained in header library unistd.h. Child processes is used to calculate mean, median and standard deviation . Before the creation of the child process program stores the timestamp of the start time. Parent process waits for child process to terminate and stores the timestamp of end. The difference between end time and start time gives the time elapsed to calculate mean, median and standard deviation. Parent and child process communicate using IPC POSIX shared memory. Program uses shm_open( ) function to create shared memory for parent and child process and ftruncate( ) function to allocate the memory space. mmap( ) establishes the memory-mapped file containing the shared memory object. It returns a pointer to memory mapped file which is used to access the shared memory object.The child process reads and outputs the contents of the shared memory. Process calls shm_unlink( ) function to remove the shared memory space.

In Assgn2-ThStat-CS17BTECH11018 program created three thread of process by using pthread_create( ) function. Those threads are used to mean, median and standard deviation. Before the creation of the thread timestamp is stored and after the joining the thread timestamp is stored. The difference between the start time and end time gives us the time taken the process to calculate mean, median and standard deviation. The parent process waits for all the threads to complete their execution and join to parent processs.

Code Documentation

Assgn-ProcStat-CS17BTECH11018. Cpp

In program creates the name of the shared memory object “/process” then takes the input all the array elements in array pointer.

SIZE is amount of memory you want to allocate.

shm_open( ) create / open POSIX shared memory object.

int shm_open(const char *name, int oflag, mode_t mode); takes arguments such as name of the shared memory and oflag is a bit masked which takes only O_RDONLY and O_RDWR .

Argument mode_t takes value 0666, means all the programs can read and write but can't execute.

mmap( ) creates a new mapping from virtual address space of calling process.

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);

If Argument addr is NULL , then kernel choses the address at which to create the mapping.

Length argument specifies the length of the mapping.

Prot argument describes the desired memory protection of the mapping, in this program PROT_READ // pages may be read

PROT_WRITE //page may be written

MAP_SHARED Share this mapping. Updates to the mapping are visible to other processes mapping the same region are carried through to the underlying file.

shm_unlink( ) unlink shared memory object.

int shm_unlink(const char *name);

Name takes the name of the shared memory object to unlink.

Assgn-ThStat-CS17BTECH11018.cpp

In program defines the thread by using "p_thread" then takes the input of the array stored in the pointer.

pthread_create( ) - creates a new thread

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

Pthread_t - takes the thread id

Pthread_atttr_t - The attr argument points to a pthread_attr_t structure whose contents are used at thread creation time to determine attributes for the new thread; this structure is initialized using pthread_attr_init(3) and related functions. If attr is NULL, then the thread is created with default attributes.

Start routine( ) - The pthread_create() function starts a new thread in the calling process. The new thread starts execution by invoking start_routine(); arg is passed as the sole argument of start_routine().

pthread_join( ) - joins with a thread

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

The pthread_join() function waits for the thread specified by thread to terminate. If that thread has already terminated, then pthread_join() returns immediately. The thread specified by thread must be joinable.

If retval is not NULL, then pthread_join() copies the exit status of the target thread (i.e., the value that the target thread supplied to pthread_exit(3)) into the location pointed to by retval. If the target thread was canceled, then PTHREAD_CANCELED is placed in the location pointed to by retval.

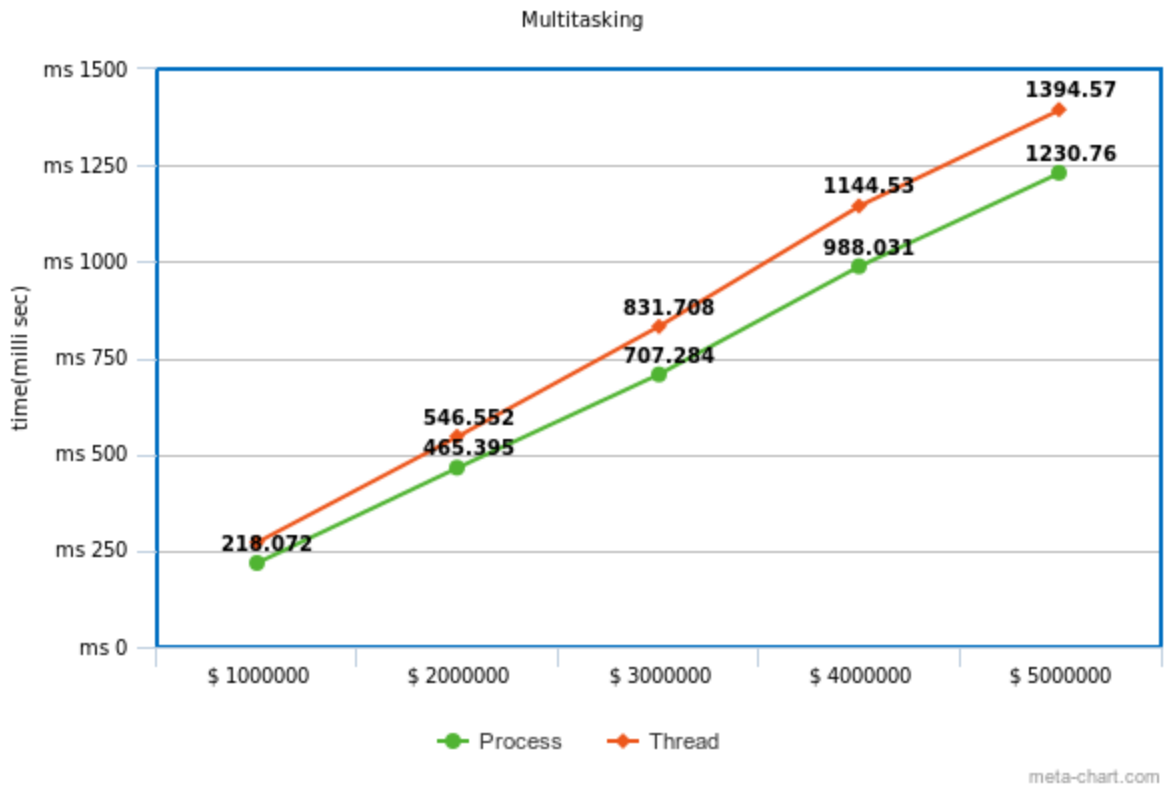
DATA

The value of the element of the array given here are random number between 1 to 100.

No of threads (10 ^ 6)	Multiprocessing (ms)	Multitasking (ms)
1	218.072	271.497
2	465.395	546.552
3	707.284	831.708
4	988.031	1144.53

5	1230.76	1394.57

RESULTS



The values of the multithreading is larger because we didn't had any inter - process communication and in which multiprocessing is taking less time.

Analysis

Multiprocessing and multithreading both increases the performance of the system. **Multiprocessing** is adding more number of or CPUs/processors to the system which increases the computing speed of the system. **Multithreading** is allowing a process to create more threads which increase the responsiveness of the system. Multithreading increases the **responsiveness** as if one thread of a process is blocked or performing the lengthy operation, the process still continues. The second benefit of multithreading is **resource sharing** as several threads of a process share same code and data within the same address space.

Usually multithreading is faster as its creation is faster and economical too but that depends on the task we are computing.

If the various task are large separate with only occasional communication between them, then multiple processes offers the advantage of being able to split the processes across different compute servers.

Here while calculating we didn’t have done any interprocess communication between threads and process hence multiprocessing is taking less time.