20/09/19

Threads :-

- Smallest transferable unit b/w the cores.
- Thread - light weight process.
- Process without any threads - Heavy weight
- Each process have multiple threads. ∴ thread

- Creating / terminating thread takes less time than a process.
- All the global data in all the threads of the same process is shared.

* • Thread id - unique id for each thread.
- Prog Counter - Each thread uses the same code so code is shared.
  - So each thread has a prog. counter to tell which part of the code it will run.
- Reg set, stack pointer - Diff. for diff. threads.
- Local variable - Diff. for each thread. & Return addresses
- Signal Handler - Global for all the threads of the process. & signal
- Signal mask - Diff./local for each thread.
  - Tells the thread which signal it needs to respond to.
- Priority - Diff for each thread
- Return value - errno. - Diff. for each thread.

* Instruction - Process inst. like inst. from OS or i/o are global to all thread but each thread have diff. mask for diff. instruction.

- To use the capability of multi-core processors the program should be written with parallel
- In any text editor – taking i/p from keyboard one thread and displaying o/p on the screen is another thread.
- Diff threads can go for i/o independently going for i/o that thread will be blocked/ suspended until i/o is completed.

* Return Addr. – The addr where ~~code~~ control should go after a thread has completed execution.
  – Diff for diff threads.

* User Level Thread :– Diff from kernel level thread
  – Can be unlimited in no. Kernel will never know the no. of user level threads.
  – Thread will not keep a list of created threads and it will not know who created it. So no mechanism of parent/child thread.
  – A thread can join any thread. Not necessarily to join the parent thread only.
- POSIX Pthreads, JAVA threads, WIN 32 threads – Libraries used to create user level threads.
  – Management of user level threads is not done

by the kernel. Management is done by the user application.

- kernel space is not allocated to user level thread.

★ Kernel Level Threads -

- Limited in no. because it uses resources of kernel space.
- Slow compared to user threads.
- Scheduling is done by kernel level threads.
- All user level threads should be mapped with kernel level threads.

• Many-to-One Mapping - Many user thread mapped to one kernel thread.

• One-to-one Mapping - Each user thread creates a kernel level thread and that kernel thread is mapped to kernel space.

- No. of user thread = kernel thread.

• Many-to-many Mapping - Many user thread is mapped to many kernel thread.

- User threads are always greater than no. of kernel threads.

- Combined Mapping – Both many-to-many and one-to-one mapping is used for diff. purpose.
  - Eg :- Solaris 8 and earlier, IRIX, HP-UX, Tru64 UNIX.
- CPU decides kernel OS decides which kernel thread will execute and that kernel thread will then decide which user thread will execute on it's behalf.

* Many-to-one – Any one user thread blocking (if want for I/O) will result in that kernel thread blocking so all other user threads are also blocked.

* One-to-one – Most commonly used today.
  - Gives maximum concistancy
  - Eg :- Windows, Linux, etc
  - No. of threads per process may be restricted.
  - More concurrency than many-to-one.

* Many-to-many – If any one kernel thread is blocked still other kernel thread is unblocked so user threads can be executed using those unblocked kernel thread.

- Two-level model – Combined mapping.
  - Imp: threads are one-to-one mapped and other are many-to-many mapped.
  - Used in systems with decent resource constraints.

* Relationship b/w Threads & Process :–

  * 1:1 (Thread : Process) – Each thread is a unique process with it's own

  Eg :–

  * 1:M – Front function – one function can be part of multiple classes – OOP.
    - One thread may be a part of more than 1 process.
    - Eg :- Ra (Clouds), Emerald.
  * M:M – Combination of 1:M & M:1.

23/09/19

Page No.:
Date: / /

★ ~~Benefits of Threads.~~

* A new thread that starts its execution from a specific function which has to be specified while creating the thread.

- Thread function has to be void * function with void * argument.
- void * is generic data type which can be type casted to any data type.

• pthread_create (&tid, &attr, &runner, argv[1]);

- &tid - To get the thread id of the new thread we pass the addr of a variable where the tid will be stored.
- &attr - Pass the addr. of the space of this struct which contains attributes like scheduling alg., priority, etc.
- runner - The function which shall be executed by the thread.
- Function name is the addr. of the function code space.
- argv[1] - Arguments req. by the function 'runner' to run.

• gcc -o exe threads.c -lpthread.

Dynamically linked this library which includes functions for threads.

execution from
to be specified

d function with
ch can be type

name, arg[i]);

new thread
ve the tid

d this struct
scheduling

will be
d.
n code space.
ion 'name'

ated this
includes
threads.

— To pass multiple arguments to the function, store the arguments in a structure and pass the address as void *.

• pthread_attr_init(&attr);
— Initialise the attribute to default value & change only the specific attributes.
— # include<pthread.h>.

* Balancing of core — Cores cannot be balanced for equal workload because execution time can be computed only after the execution of the prog.
— so uses dynamic balancing.

* Data splitting — Matrix multiplication — data split b/w diff. cores.

* Data Parallelism — ~~Data is given to all the cores and~~ same data is given to multiple cores for diff. task.

* Task Parallelism — same data is given to diff. cores for diff tasks.

• pthread_join(tid, NULL);
— Similar as wait pid.
— wait for thread with specific tid.

— If pthread_create() & pthread_join() are consecutively then there is no parallelism because parent is waiting for the thread to complete.

- In linux, a new thread will always create a new process internally. — Global data is shared and file descriptors are also shared.
— getpid() — This will always give the same pid of the parent which created the thread and not the pid of new process created because of the thread.
— It displays the thread group pid which is the main pid which created the thread.
— To get the actual pid of new process we need to use a system call.
— The new process will only have it's local stack, rest all will be shared.
— pthread_create / _join will use the same system call do_fork() but with diff arguments to create/wait for a thread in linux.
- More than 1 thread can be created using the same function. The threads will run concurrently to each other.
- Race condition is possible with shared variables.

24/09/19 If exec() is executed in a thread then the thread group leader process (the process which created it) will die.

study time
Page No.:
Date: / /

- To avoid race condition we can create a thread, execute it completely, and then create another thread.

- If a newly thread creates another thread, then both the threads will have the same pid viz. the group leader process id only.

- Os already have threads created but not executing, so when a process creates a thread Os just fills it with req. structure and starts executing - makes system faster & efficient.

* Thread Creating -

+ pthread_create - 0 is returned on successful creation.

- On non-successful operation, non-zero -ve value is returned.

+ Attributes :-

```
# define __SIZEOF_PTHREAD_ATTR_T 56
typedef union
{
  char __size[__SIZEOF
  long int __align;
} pthread_attr_t;
```

- All attributes of a thread is stored in this structure

- extern int pthread_attr_init(pthread_attr_t *);
  — Initializes the attribute with default values.

  — Once a thread is created, any changes done in the structure is not reflected in the already created thread.
  — Same structure can be used for multiple threads.

- pthread_attr_destroy — To destroy the attr structure.

* Thread States :-

- Joinable Thread — A thread releases all the resources when any other thread joins it.

  — By default a thread is joinable.

- Detachable Thread — A thread releases all the resources immediately when it is exited using pthread_exit().

  — For A joinable thread has to link to other thread to release the resources even if it has finished execution.
  — Only one thread can join some other thread.
  — Any thread can join any other thread from that same pid process.

— The state of the thread can be changed any no. of times.

* Scheduling policy :—

— 3 diff. sched algorithms—
i) FIFO ii) Round Robin iii) Others.
:— Real time —Real time. —Non Realtime

— Total no priorities available — 0 to 139.
— 0-99 — Real Time Priority
— 100-139 — Non real time priority.

— Round Robin & FIFO are implemented using the same data structure.
— Round Robin with ∞ time quantum is FIFO.

— In Round Robin a new process joins at the back of the queue and in FIFO a process joins at the front of the queue.

— So in FIFO the same process keeps on executing until the process is executed completely.

— A non-real time thread will always have a non-real time ∴ algorithm so no need to change the policy.

— These algorithms are only for processes in the same priority no.

Page No.: study time
Date: / /

- For real time threads/processes the priority value need also be set between 0 and 99.
- All these are done using attribute functions.