Embedded Operating Systems

Agenda

- Synchronization
- Semaphore vs Mutex
- Deadlock
- Multi-threading

Semaphore

- Refer yesterday's notes.

Mutex

- Mutex is used to ensure that only one process can access the resource at a time.
- Functionally it is same as "binary semaphore".
- Mutex can be unlocked by the same process/thread, which had locked it.

Semaphore vs Mutex

- S: Semaphore can be decremented by one process and incremented by same or another process.
- M: The process locking the mutex is owner of it. Only owner can unlock that mutex.
- S: Semaphore can be counting or binary.
- M: Mutex is like binary semaphore. Only two states: locked and unlocked.
- S: Semaphore can be used for counting, mututal exclusion or as a flag.
- M: Mutex can be used only for mutual exclusion.

Critical Section

- Section (block) of a code that should be executed by single process at a time. If multiple processes/threads execute it concurrently, then unexpected results may happen.

- Example: Doubly linked list -- add node at position

```
// create new node (nn)
// trav till position - 1 (trav)
// get address of next node (temp)
// add node between trav and temp
nn->next = temp;
nn->prev = trav;
trav->next = nn;
temp->prev = nn;
```

- Critical section problem can be solved using mutex.

```
init(mutex);
// ...
lock(mutex);
// ...
// ... critical section
// ...
unlock(mutex);
// ...
```

Deadlock

- Deadlock occurs when four conditions/characteristics hold true at the same time.
  - No preemption: A resource should not be released until task is completed.
  - Mutual exclusion: Resources is not sharable.
  - Hold & Wait: Process holds a resource and wait for another resource.
  - Circular wait: Process P1 holds a resource needed for P2, P2 holds a resource needed for P3 and P3 holds a resource needed for P1.

Deadlock Prevention

- OS syscalls are designed so that at least one deadlock condition does not hold true.
- In UNIX multiple semaphore operations can be done at the same time.

```
P(s1);
P(s2);
```

is same as

```
P(s1, s2);
```

- P(s1, s2) is atomic operation.

Deadlock Avoidance

- Processes declare the required resources in advanced, based on which OS decides whether resource should be given to the process or not.
- Algorithms used for this are:
  - Resource allocation graph: OS maintains graph of resources and processes. A cycle in graph indicate circular wait will occur. In this case OS can deny a resource to a process.
  - Banker's algorithm: A bank always manage its cash so that they can satisfy all customers.
  - Safe state algorithm: OS maintains statistics of number of resources and number processes. Based on stats it decides whether giving resource to a process is safe or not (using a formula):
    - Max num of resources required < Num of resources + Num of processes
      - If condition is true, deadlock will never occur.
      - If condition is false, deadlock may occur.

Multi-Threading

Thread concept

- Threads are used to execute multiple tasks concurrently in the same program/process.

- Thread is a light-weight process.
  - For each thread new control block and stack is created. Other sections (text, data, heap, ...) are shared with the parent process.
  - Inter-thread communication is much faster than inter-process communication.
  - Context switch between two threads in the same process is faster.
- Thread stack is used to create function activation records of the functions called/executed by the thread.

Process vs Thread

- In modern OS, process is a container holding resources required for execution, while thread is unit of execution/scheduling.
- Process holds resources like memory, open files, IPC (e.g. signal table, shared memory, pipe, etc.).
- PCB contains resources information like pid, exit status, open files, signals/ipc, memory info, etc.
- CPU time is allocated to the threads. Thread is unit of execution.
- TCB contains execution information like tid, state, scheduling info (priority, sched algo, time left, ...), Execution context, Kernel stack, etc.
- terminal> ps -e -o pid,nlwp,cmd
- terminal> ps -e -m -o pid,tid,nlwp,cmd
- terminal> cat /proc/pid/maps

main thread

- For each process one thread is created by default called as main thread.
- The main thread executes entry-point function of the process.
- The main thread use the process stack.
- When main thread is terminated, the process is terminated.
- When a process is terminated, all threads in the process are terminated.