ICS 143 - Principles of Operating Systems

Operating Systems - Review

Prof. Nalini Venkatasubramanian

nalini@ics.uci.edu

What is an Operating System?

An OS is a program that acts an intermediary between the user of a computer and computer hardware.

Major cost of general purpose computing is software.

OS simplifies and manages the complexity of running application programs efficiently.

Operating System Views

Resource allocator

to allocate resources (software and hardware) of the computer system and manage them efficiently.

Control program

Controls execution of user programs and operation of I/O devices.

Kernel

The program that executes forever (everything else is an application with respect to the kernel).

Operating System Spectrum

- Monitors and Small Kernels
- Batch Systems
  - Polling vs. interrupt
- Multiprogramming
- Timesharing Systems
  - concept of timeslice
- Parallel and Distributed Systems
  - symmetric vs. asymmetric multiprocessing
- Real-time systems
  - Hard vs. soft realtime

Computer System Structures

- Computer System Operation
- I/O Structure
- Storage Structure
  - Storage Hierarchy
- Hardware Protection
- General System Architecture
- System Calls and System Programs
- Command Interpreter

Operating System Services

- Services that provide user-interfaces to OS
  - Program execution - load program into memory and run it
  - I/O Operations - since users cannot execute I/O operations directly
  - File System Manipulation - read, write, create, delete files
  - Communications - interprocess and intersystem
  - Error Detection - in hardware, I/O devices, user programs
- Services for providing efficient system operation
  - Resource Allocation - for simultaneously executing jobs
  - Accounting - for account billing and usage statistics
  - Protection - ensure access to system resources is controlled

Process Management

Process - fundamental concept in OS

- Process is a program in execution.
- Process needs resources - CPU time, memory, files/data and I/O devices.

OS is responsible for the following process management activities.

- Process creation and deletion
- Process suspension and resumption
- Process synchronization and interprocess communication
- Process interactions - deadlock detection, avoidance and correction

Process Concept

- An operating system executes a variety of programs
  - batch systems - jobs
  - time-shared systems - user programs or tasks
  - job and program used interchangeably
- Process - a program in execution
  - process execution proceeds in a sequential fashion
- A process contains
  - program counter, stack and data section
- Process States
  - e.g. new, running, ready, waiting, terminated.

Process Control Block

Contains information associated with each process

- Process State - e.g. new, ready, running etc.
- Program Counter - address of next instruction to be executed
- CPU registers - general purpose registers, stack pointer etc.
- CPU scheduling information - process priority, pointer
- Memory Management information - base/limit information
- Accounting information - time limits, process number
- I/O Status information - list of I/O devices allocated

Schedulers

Long-term scheduler (or job scheduler) -

- selects which processes should be brought into the ready queue.
- invoked very infrequently (seconds, minutes); may be slow.
- controls the degree of multiprogramming

Short term scheduler (or CPU scheduler) -

- selects which process should execute next and allocates CPU.
- invoked very frequently (milliseconds) - must be very fast

Medium Term Scheduler

- swaps out process temporarily
- balances load for better throughput

Process Creation

Processes are created and deleted dynamically

Process which creates another process is called a *parent* process; the created process is called a *child* process.

Result is a tree of processes

e.g. UNIX - processes have dependencies and form a hierarchy.

Resources required when creating process

CPU time, files, memory, I/O devices etc.

Process Termination

Process executes last statement and asks the operating system to delete it (*exit*).

- Output data from child to parent (via wait).
- Process' resources are deallocated by operating system.

Parent may terminate execution of child processes.

- Child has exceeded allocated resources.
- Task assigned to child is no longer required.
- Parent is exiting
  - OS does not allow child to continue if parent terminates
  - Cascading termination

Threads

Processes do not share resources well

- high context switching overhead

A thread (or lightweight process)

- basic unit of CPU utilization; it consists of:
  - program counter, register set and stack space

A thread shares the following with peer threads:

- code section, data section and OS resources (open files, signals)

Collectively called a task.

Heavyweight process is a task with one thread.

Thread support in modern systems

User threads vs. kernel threads, lightweight processes

1-1, many-1 and many-many mapping

Producer-Consumer Problem

Paradigm for cooperating processes;

producer process produces information that is consumed by a consumer process.

We need buffer of items that can be filled by producer and emptied by consumer.

- Unbounded-buffer places no practical limit on the size of the buffer. Consumer may wait, producer never waits.
- Bounded-buffer assumes that there is a fixed buffer size. Consumer waits for new item, producer waits if buffer is full.

Producer and Consumer must synchronize.

Interprocess Communication (IPC)

- Mechanism for processes to communicate.
  - Via shared memory
  - Via message-passing: processes communicate without resorting to shared variables.
- IPC via shared memory
  - An area of memory shared among the processes that wish to communicate
  - The communication is under the control of the processes not the operating system.
  - Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.

Interprocess Communication (IPC)

- IPC via message-passing
  - **send**(*message*)
  - **receive**(*message*)
  - Direct vs. indirect communication
  - The message size is either fixed or variable.

CPU Scheduling

Scheduling Objectives

Levels of Scheduling

Scheduling Criteria

Scheduling Algorithms

Multiple Processor Scheduling

Real-time Scheduling

Scheduling Policies

FCFS (First Come First Serve)

- Process that requests the CPU *FIRST* is allocated the CPU *FIRST.*

SJF (Shortest Job First)

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.

Priority

- A priority value (integer) is associated with each process. CPU allocated to process with highest priority.

Round Robin

- Each process gets a small unit of CPU time

MultiLevel

- ready queue partitioned into separate queues
- Variation: Multilevel Feedback queues.

Process Synchronization

The Critical Section Problem

Synchronization Hardware

Semaphores

Classical Problems of Synchronization

Critical Regions

Monitors

The Critical Section Problem

Requirements

- Mutual Exclusion
- Progress
- Bounded Waiting

Solution to the 2 process critical section problem

Bakery Algorithm

Solution to the n process critical section problem

Before entering its critical section, process receives a number. Holder of the smallest number enters critical section.

Synchronization Hardware

Test and modify the content of a word atomically - Test-and-set instruction

```
function Test-and-Set (var target: boolean): boolean;
  begin
     Test-and-Set := target;
     target := true;
  end;
```

Mutual exclusion using test and set.

Bounded waiting mutual exclusion using test and set.

"SWAP" instruction

Mutual Exclusion with Test-and-Set

Shared data: var lock: boolean (initially false)

Process Pi

```
repeat
    while Test-and-Set (lock) do no-op;
                        critical section
            lock := false;
                        remainder section
        until false;
```

Bounded Waiting Mutual Exclusion with Test-and-Set

```
var j : 0..n-1;
      key : boolean;
repeat
     waiting [i]  := true; key := true;
    while waiting[i] and key do key := Test-and-Set(lock);
     waiting [ i ] := false;
 critical section
      j := i+1 mod n;
      while (j <> i )  and (not waiting[j]) do j := j + 1 mod n;
       if j = i  then lock := false;
                 else waiting[j] := false;
  remainder section

until false;
```

Semaphore

Semaphore *S* - integer variable

- used to represent number of abstract resources.
- Binary vs. counting semaphores.

Can only be accessed via two indivisible (atomic) operations

wait (*S*): **while** *S* <= 0 **do** no-op
S := *S*-1;

signal (S): *S* := *S*+1;

- P or wait used to acquire a resource, decrements count
- V or signal releases a resource and increments count
- If P is performed on a count <=0, process must wait for V or the release of a resource.

Block/resume implementation of semaphores

Classical Problems of Synchronization

Bounded Buffer Problem

Readers and Writers Problem

Dining-Philosophers Problem

Readers-Writers Problem

Shared Data

```
var mutex, wrt: semaphore (=1);
    readcount: integer (= 0);
```

Writer Process

```
wait(wrt);
   …
  writing is performed
  …
signal(wrt);
```

Reader process

```
wait(mutex);
   readcount := readcount +1;
   if readcount = 1 then wait(wrt);
signal(mutex);
        …
          reading is performed
        …
wait(mutex);
   readcount := readcount - 1;
   if readcount = 0 then signal(wrt);
signal(mutex);
```

Critical Regions

High-level synchronization construct

A shared variable *v* of type *T* is declared as:

var *v*: **shared** *T*

Variable v is accessed only inside statement

region *v* when *B* **do** *S*

where *B* is a boolean expression.

While statement *S* is being executed, no other process can access variable *v*.

Monitors

High-level synchronization construct that allows the safe sharing of an abstract data type among concurrent processes.

```
type monitor-name = monitor
    variable declarations
    procedure entry P1 (…);
        begin … end;
    procedure entry P2 (…);
        begin … end;
                .
                .
                .
    procedure entry Pn(…);
        begin … end;
    begin
        initialization code
    end.
```

Hoare vs. Mesa Monitors