Frequently asked questions from the previous class survey

- Processors, CPU, and Core: Can we please disambiguate?
- Cores: Too much of a good thing?
- Caches: L1, L2, L3 on the CPU?
- Why are cache hits so high? Ans: Spatial/temporal locality (Working Sets).
- Why not have only a gigantic cache and do away with Main Memory altogether?
- Is the Kernel in Main memory or Cache?
- What runs in main memory?
- Why do you need hardware timers or interrupt processing?
- How many processes does a modern processor ‘run’ at the same time?
- Quantum

Non-aligned memory addresses? Does user code have to worry about it?
- Is kernel mode like root in Linux?
- How does the kernel know how much memory to give each application?
- Which is better? Replication or improving?
- Horizontal scaling vs vertical scaling

Processors, CPU, and Core: Can we please disambiguate?
- Cores: Too much of a good thing?
- Caches: L1, L2, L3 on the CPU?
- Why are cache hits so high? Ans: Spatial/temporal locality (Working Sets).
- Why not have only a gigantic cache and do away with Main Memory altogether?
- Is the Kernel in Main memory or Cache?
- What runs in main memory?
- Why do you need hardware timers or interrupt processing?
- How many processes does a modern processor ‘run’ at the same time?
- Quantum

- Processors
- A process in memory
- Process Control Blocks
- Interrupts & Context switches
- Operations on processes
- Creation

The oldest and most important abstraction that an operating system provides
- Supports the ability to have (pseudo) concurrent operation
- Even if there is only 1 CPU
What is a process?

- A process is the **execution** of an application program with restricted rights
- It is the abstraction for protected execution provided by the kernel

All modern computers do several things at a time

- Browsing while e-mail client is fetching data
- Printing files while burning a CD-ROM

Multiprogramming

- CPU switches from process-to-process quickly
- Runs each process for 10s-100s of milliseconds

Multiprogramming and parallelism

- At any instant of time, the CPU is running only one process
- In the course of 1 second, it is working on several of them
- Gives the illusion of parallelism
  - Pseudoparallelism

A process is the unit of work in most systems

- Arose out of a need to compartmentalize and control concurrent program executions
- A process is a program in execution
- Essentially an activity of some kind
  - Has a program, input, output and a state.

A process is just an instance of a program [1/2]

- In much the same way that an object is an instance of a class in object-oriented programming
- Each program can have zero, one or more processes executing it
- For each instance of a program, there is a process with its own copy of the program in memory.
A process is just an instance of a program [2/2]

- Conceptually each process has its own virtual CPU
- In reality, the CPU switches back-and-forth from process to process
- Processes are not affected by the multiprogramming
  - Or relative speeds of different processes

An example scenario: 4 processes

- Four Program Counters
- 4 processes in memory

Example scenario: 4 processes

- At any instant only one process executes
- Viewed over a long time, all processes have made progress

Programs and processes

- Programs are passive, processes are active
- The difference between a program and a process is subtle, but crucial

Analogy of a culinary-minded computer scientist baking cake for his daughter

<table>
<thead>
<tr>
<th>Analogy</th>
<th>Mapping to real settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Birthday cake recipe</td>
<td>Program (algorithm expressed in a suitable notation)</td>
</tr>
<tr>
<td>Well-stacked kitchen</td>
<td>Input Data</td>
</tr>
<tr>
<td>flour, eggs, sugar,</td>
<td></td>
</tr>
<tr>
<td>vanilla extract, etc.</td>
<td></td>
</tr>
<tr>
<td>Computer scientist</td>
<td>Processor (CPU)</td>
</tr>
</tbody>
</table>

- Process is the activity of
  1. Baker reading the recipe
  2. Fetching the ingredients
  3. Baking the cake

SLIDES CREATED BY: SHRIDEEP PALICKARA
Scientist’s son comes in screaming about a bee sting

- Scientist records where he was in the recipe
- State of current process is saved
- Gets out a first aid book, follows directions in it

In our example, the scientist has switched to a higher priority process ...

- FROM Baking
  - Program is the cake recipe
  - To administering medical care
  - Program is the first-aid book
  - When the bee sting is taken care of
    - Scientist goes back to where he was in the baking

Key concepts

- Process is an activity of some kind; it has a
  - Program
  - Input and Output
  - State

- Single processor may be shared among several processes
  - Scheduling algorithm decides when to stop work on one, and start work on another

HOW A PROGRAM BECOMES A PROCESS

The journey from code to a becoming a process [1/2]

- Programmer types code in some high-level language
- A compiler converts that code into a sequence of machine instructions and stores those instructions in a file
  - Called the program’s executable image
- Compiler also defines any static data the program needs, along with its initial values, and includes them in the executable image

The journey from code to a becoming a process [2/2]

- To run the program, the kernel copies the instructions and data from the executable image into physical memory
- The kernel sets aside memory regions
  - The execution stack, to hold local variables during procedure calls
  - The heap, for any dynamically allocated data structures the program might need
- Of course, to copy the program into memory, the kernel itself must already be in memory, with its own stack and heap
A process in memory

- Stack: Function parameters, return addresses, and local variables
- Heap: Memory allocated dynamically during runtime
- Data: Global variables
- Text: Program code

Memory conservation
- Most operating systems reuse memory wherever possible
- The OS stores only a single copy of a program's instructions
- Even when multiple copies of the program are executed at the same time
- Even so, a separate copy of the program's data, heap, and stack are needed.

How a program becomes a process
- Allocation of memory is not enough to make a program into a process
- Must have a process ID
- OS tracks IDs and process states to orchestrate system resources

Program in memory
- Program image appears to occupy contiguous blocks of memory
- OS maps programs into non-contiguous blocks

Program in memory
- Mapping divides the program into equal-sized pieces: pages
- OS loads pages into memory
- When processor references memory on page
  - OS looks up page in table, and loads into memory

Advantages of the mapping process
- Allows large logical address space for stack and heap
  - No physical memory used unless actually needed
- OS hides the mapping process
  - Programmer views program image as logically contiguous
  - Some pages may not reside in memory
Finite State Machine

- An initial state
- A set of possible input events
- A finite number of states
- Transitions between these states
- Actions

How does the OS track processes?

- Via a data structure called the process control block, or PCB
- The PCB stores all the information the OS needs about a particular process
  - Where it is stored in memory, where its executable image resides on disk, which user asked it to execute, what privileges it has, etc.
- The set of the PCBs defines the current state of the OS

Where is the PCB stored?

- Since PCB contains the critical information for the process
  - It must be kept in an area of memory protected from normal user access
  - Maintained in kernel memory

Process state transition diagram: When a process executes it changes state

- New
- Admitted
- Ready
- Running
- Exit admitted
- Exit
- Exit failed
- Exit
- Dead
- I/O or event wait
- I/O or event completion
- Waiting
- Scheduler dispatch
- Terminated
- Exit

Each process is represented by a process control block (PCB)

- PCB is a repository for any information that varies from process to process.

An example of CPU switching between processes
What can be at the user level, should be.

- Allow user programs to create and manage their own processes
- If creating a process is something a process can do, then anyone can build a new version of any of these applications
  - Without recompiling the kernel or forcing anyone else to use it
- Instead of a single program that does everything, we can create specialized programs for each task, and mix-and-match what we need
  - There’s an app for that!

Interrupts and Contexts

- Interrupt causes the OS to change CPU from its current task to run a kernel routine
- Save current context so that suspend and resume are possible
- Context is represented in the PCB
  - Value of CPU registers
  - Process state
  - Memory management information

Factors that impact the speed of the context switch

- Memory speed
- Number of registers to copy
- Special instructions for loading/storing registers
- Memory management: Preservation of address space
OPERATIONS ON PROCESSES

Processes execute concurrently.
Can be created and deleted dynamically.

Process Creation: A process may create new processes during its execution

- Parent process: The creating process
- Child process: New process that was created
  - May itself create processes: Process tree
- All processes have unique identifiers

Example: Process tree in Solaris

Processes in UNIX

- init: Root parent process for all user processes
  - Get a listing of processes with `ps` command
    - `ps`: List of all processes associated with user
    - `ps -a`: List of all processes associated with terminals
    - `ps -A`: List of all active processes

Resource sharing between a process and its subprocess

- Child process may obtain resources directly from OS
- Child may be constrained to a subset of parent’s resources
  - Prevents any process from overloading system
- Parent process also passes along initialization data to the child
  - Physical and logical resources

Parent/Child processes: Execution possibilities

- Parent executes concurrently with children
- Parent waits until some or all of its children terminate
Parent/Child processes:
Address space possibilities

- Child is a duplicate of the parent
  - Same program and data as parent

- Child has a new program loaded into it

Process creation in UNIX

- Process created using `fork()`
  - `fork()` copies parent's memory image
  - Includes copy of parent's address space

- Parent and child continue execution at instruction after `fork()`
  - Child: Return code for `fork()` is 0
  - Parent: Return code for `fork()` is the non-ZERO process-ID of new child

Simple example:

```c
#include <stdio.h>
#include <unistd.h>

int main(void) {
    int x;
    x=0;
    fork();
    x=1;
    return 0;
}
```

Both parent and child execute this after returning from `fork()`

Another example

```c
#include <stdio.h>
#include <unistd.h>

int main () {
    printf("Hello World\n");
    fork();
    printf("Hello World\n");
}
```

Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
What happens when `fork()` fails?

- No child is created
- `fork()` returns -1 and sets `errno`
  - `errno` is a global variable in `errno.h`