NUS
National University
of Singapore

Processes

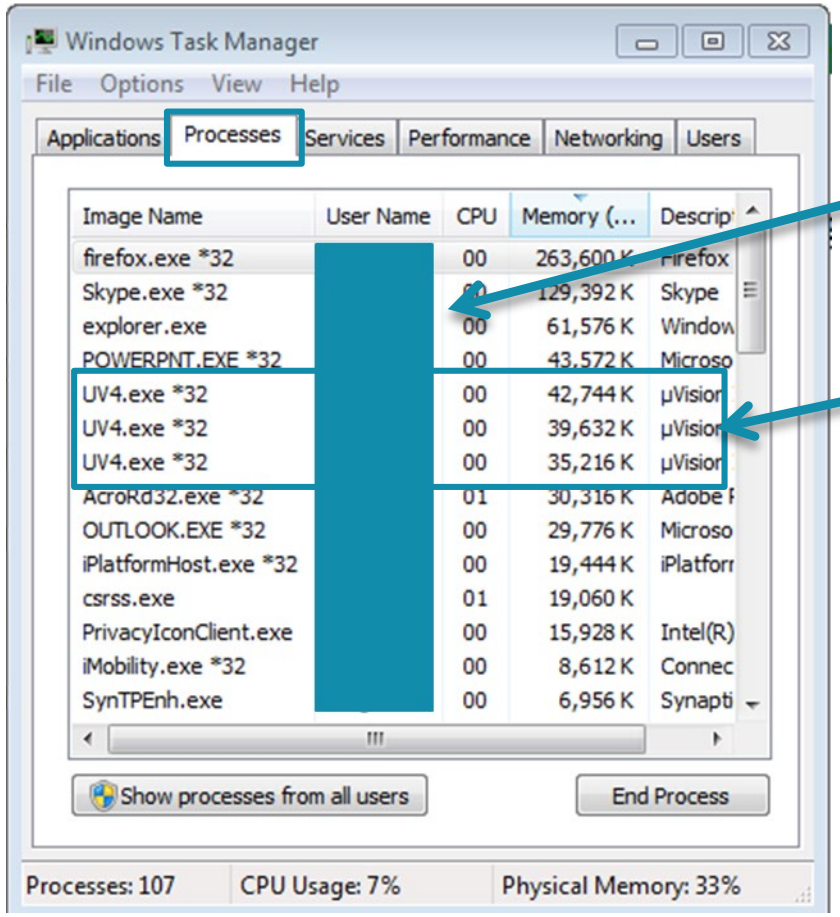
Operating System Principles

Ravi Suppiah
Lecturer, NUS SoC

Lecture Contents

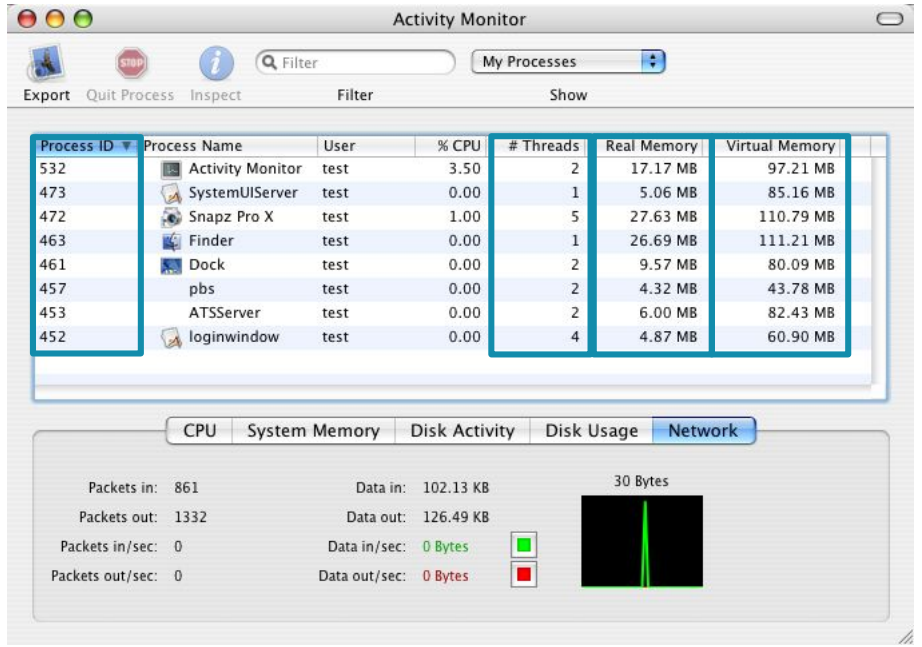
- Process– What is it
- Memory lay out
- Switching between Processes
- RTX Examples
- Process Termination
- Context Switching and States

An Instance of a Running Program



Your user name or SYSTEM/LOCAL SERVICE/NETWORK SERVICE

Multiple instances of the µVision4
Independent memory for each process
"Memory(Private Working Set)"



Try to explore the task manager, activity monitor, or similar utility on your favourite OS

An Instance of a Running Program

NUS National University of Singapore

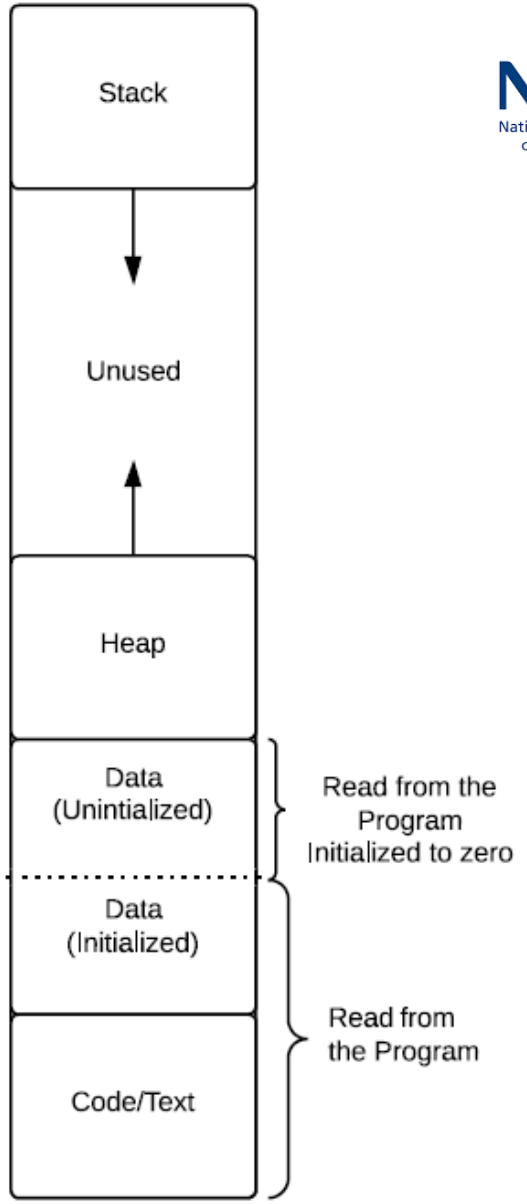
- A process can have
  - A CPU time allocation (virtual CPU, the role of OS)
  - Memory (real or virtual)
  - Process ID
  - Threads

- Are they aware of the existence of other processes?
  - The operating system's role is to create an "illusion" that a process has all it needs to be executed
  - An abstraction of hardware resources
  - Each process sees one dedicated processor and one segment of memory (although they are often shared with others)
  - But they can be aware of each other – inter-process communication (IPC)

ARM

Memory Layout of an Executing Program

- Code or Text
  - Binary instructions to be executed
  - A clone of the program
  - Usually read-only
  - Program counter (PC), points to the next instruction
- Static Data
  - Global/Constant/Static variables - shared between threads
  - If not initialized by program, will be zero or null pointer
- Heap
  - malloc/free
- Stack
  - Used for procedure calls and return
  - Stack Pointer(SP)
  - FILO (First In, Last Out)



Process – The Abstraction

NUS National University of Singapore

- Switching between executions means the operating system has to keep track of all the execution context
- Includes:
  - Memory State (code, data, heap and stack)
  - CPU state (PC, SP and other registers)
  - Also the OS state
- Hence the abstraction of the process
- Program usually refers to the instructions that are stored on disk
- Process is the program with execution context
- Some OSs may use the term "task", particularly in an embedded system context. We will use both terms interchangeably for this course
- Thread: a lightweight process; a process may have multiple threads which share the same system resources - faster creation, termination, switching and communication

ARM

Process Control Block

- The Process Control Block (PCB) or Task Control Block (TCB) maintains all the relevant information for the process:
  - Process ID
  - Process state
  - PC, SP and other registers (stored)
  - Scheduling information (priority)
  - Memory management information
  - Accounting information
  - User information
  - Inter-Process Communication (IPC)
  - Other information

- The ID points to the entry in the process table where the pointer to the PCB is stored

RTX Example

```
typedef struct OS_TCB {
 /* General part: identical for all implementations.                */
 U8    cb_type;              /* Control Block Type                   */
 U8    state;                /* Task state                           */
 U8    prio;                 /* Execution priority                   */
 U8    task_id;              /* Task ID value for optimized TCB access */
 struct OS_TCB *p_lnk;       /* Link pointer for ready/sem. wait list */
 struct OS_TCB *p_rlnk;      /* Link pointer for sem./mbx lst backwards */
 struct OS_TCB *p_dlnk;      /* Link pointer for delay list          */
 struct OS_TCB *p_blnk;      /* Link pointer for delay list backwards */
 U16   delta_time;           /* Time until time out                  */
 U16   interval_time;        /* Time interval for periodic waits     */
 U16   events;               /* Event flags                          */
 U16   waits;                /* Wait flags                           */
 void  **msg;                /* Direct message passing when task waits */
 struct OS_MUCB *p_mlnk;     /* Link pointer for mutex owner list    */
 U8    prio_base;            /* Base priority                        */
 U8    ret_val;              /* Return value upon completion of a wait */

 /* Hardware dependant part: specific for CM processor              */
 U8    ret_upd;              /* Updated return value                 */
 U16   priv_stack;           /* Private stack size, 0= system assigned */
 U32   tsk_stack;            /* Current task Stack pointer (R13)     */
 U32   *stack;               /* Pointer to Task Stack memory block   */

 /* Task entry point used for uVision debugger                      */
 FUNCP  ptask;               /* Task entry address                   */
} *P_TCB;
```

- We will come back to this later
- Check with the source code yourself - not really as intimidating as you might expect!

 <<Your Keil Directory>>\ARM\RL\RTX\SRC\CM\rt_TypeDef.h

Process Creation

NUS National University of Singapore

- By initialization, or by request of a process.
- Unique ID for the process
- Allocate memory for the PCB and other control structures (kernel) and user memory
- Initialize the PCB and memory management
- Link the PCB in the queue (see later)

ARM

Process Termination

NUS National University of Singapore

- Stopped by the OS/user (why?) or terminate itself

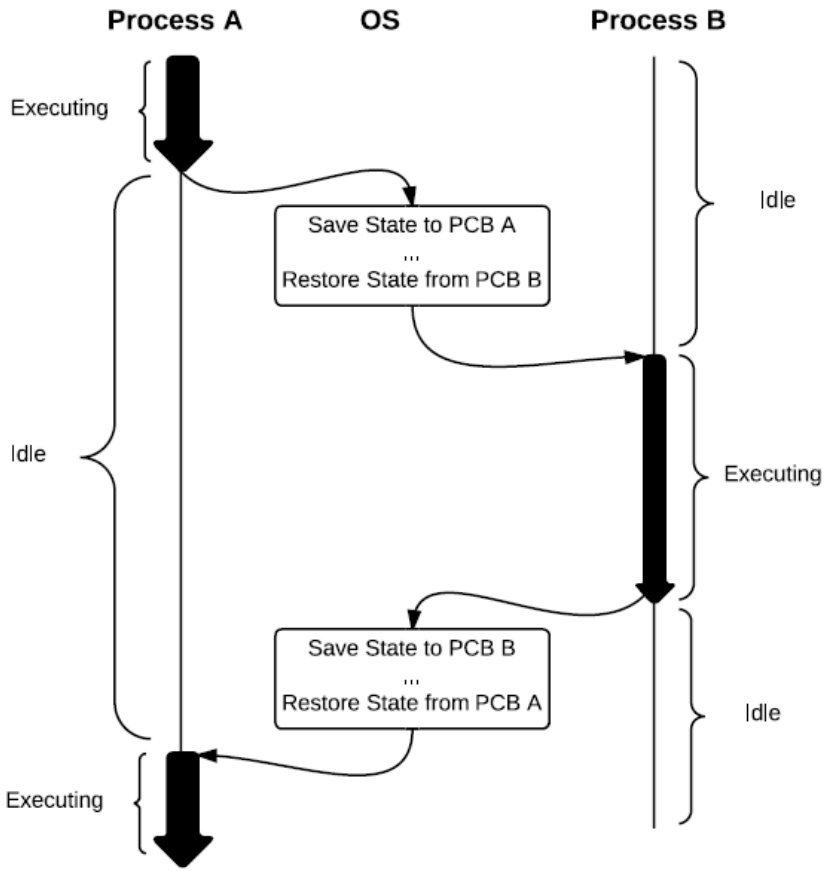
- Handle the output of the process
- Release the resources and reclaim the memory
- Unlink the PCB

- In embedded software, some processes may never terminate. Terminating implies a fault.

ARM

Context Switching

- The PCB makes context switching a bit easier
  - Scheduler will start or stop a process accordingly
  - Stores necessary information in the PCB to stop
    - Hardware registers
    - Program Counter
    - Memory states, stack and heap
    - State
  - Similarly, loads necessary information from the PCB
- Notice that context switching does consume time!
  - Could be up to several thousand CPU cycles
  - Overhead and bottleneck
  - Hardware support is also needed



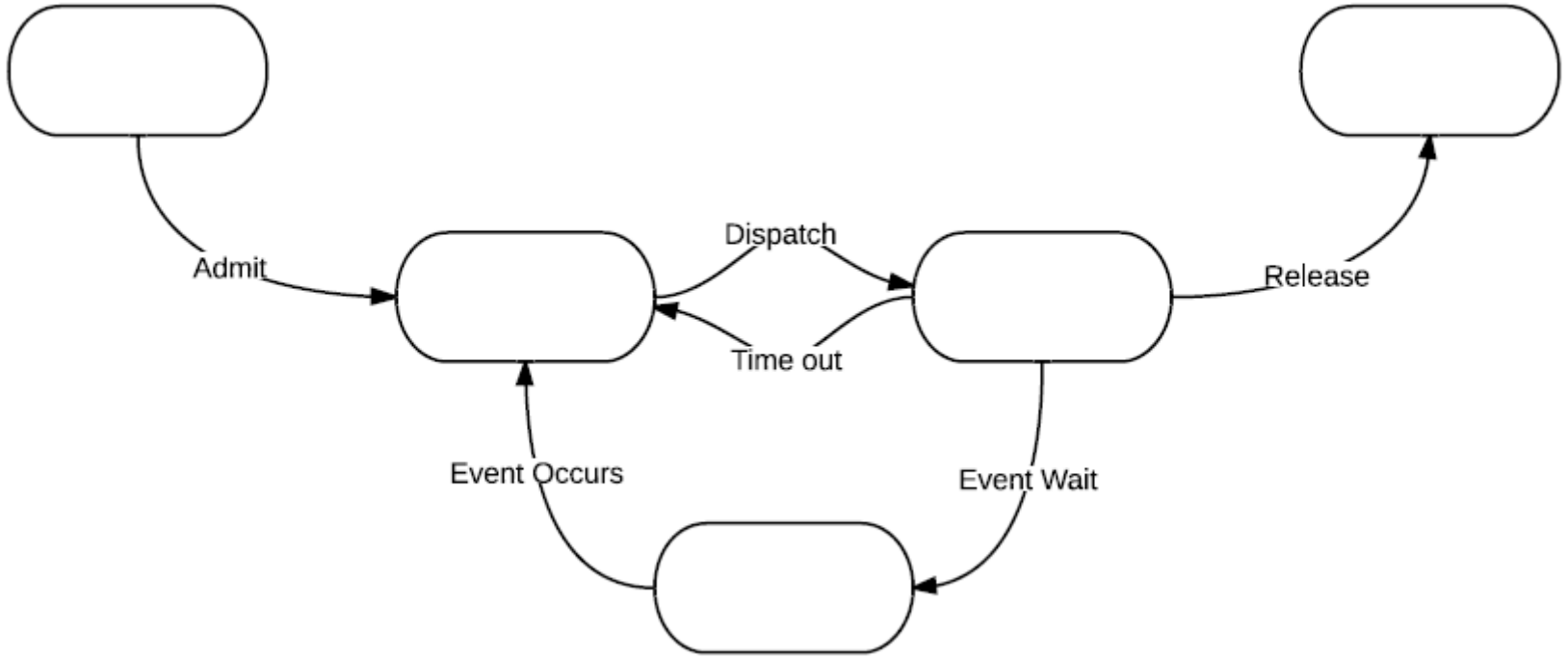
Process States

NUS National University of Singapore

- Different process states during the lifetime cycle
- Many variants but a standard model has five states:
  - New: just been created, not ready for the queue
  - Ready: can be loaded by the OS
  - Running: scheduler has picked this process from the queue and executed it, usually only one
  - Blocked: not in the queue, waiting
  - Exit: finished, needs to terminate

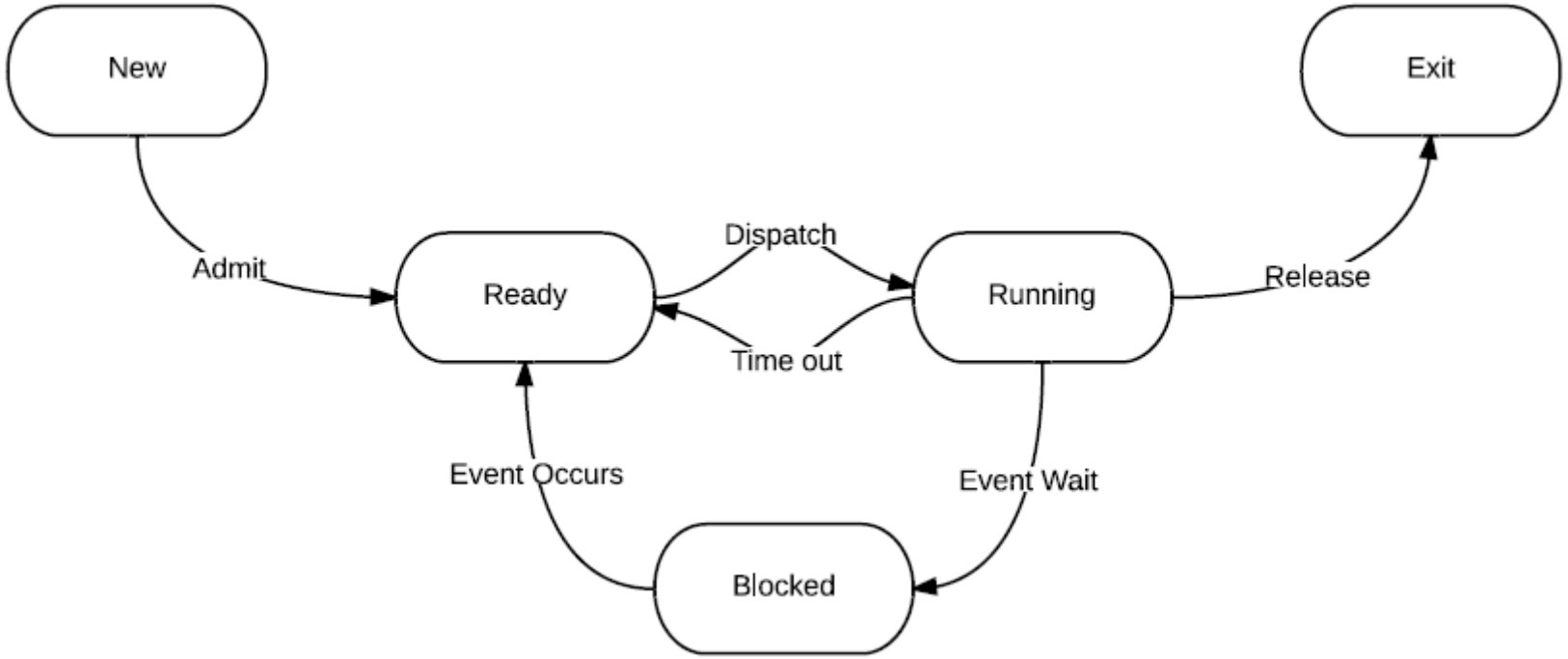
ARM

State Transition



- State transition as a result of OS scheduling, external interrupts or program requests
- Try to fill in the states in each block

State Transition



- Admit: process is fully loaded into memory and control is established
- Dispatch: scheduler assigns CPU to the process
- Time out: expired or preempted, pushed back to the queue
- Event Wait/Event Occurs: generally requests that cannot be met at the moment, has to wait until something occurs
  - OS not ready for a service
  - Unavailable resource
  - Wait for an input
- Release: release resources and end the process

Process State

- State information is also recorded by the PCB
- Context switch takes place whenever a process leaves/enters the running state
- Processes may make a transition voluntarily or involuntarily, e.g., end the program vs error
- OS typically maintains queue or queue-like (list) structures for processes in the same states (many pointers in the PCB)
  - RTX: rt_list.c

RTX Example

```
/* Values for 'state'   */
#define INACTIVE      0
#define READY         1
#define RUNNING       2
#define WAIT_DLY      3
#define WAIT_ITV      4
#define WAIT_OR       5
#define WAIT_AND      6
#define WAIT_SEM      7
#define WAIT_MBX      8
#define WAIT_MUT      9
```

Variants of blocked States: The name indicates the event to invoke the process

- No Exit state
- Embedded software don't really enjoy the concept of termination
- More on this in later lectures and labs

The End!

NUS National University of Singapore

- Next, we will look at Scheduling!

ARM