UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

SE350 RTX LAB 1 - 2

P1B Processor Management

Irene Yiqing Huang
Electrical and Computer Engineering Department

Copyright by Yiqing Irene Huang 2022. All rights reserved

Reading the Lab Manual

Section	Topics	
10.2	Cortex-M3 Processor	Review
10.4	Exceptions and Interrupts	Review
9.1-9.2	The Tumb-2 ISA and AAPCS	Skim
9.5	SVC Programming	Review
5.1	RTX P1	Study/Review
5.2	Demonstration	Study
5.3	Third-party testing framework	Study
1.1-1.2	Introduction of the RTX	Review
2.3	Processor Management	Study
6.1	Processor Management FAQ	Skim

Copyright by Yiqing Irene Huang 2022. All rights reserved

P1B Overview

- A multiprogramming kernel
  - Fixed number of co-existing processes
- Scheduling
  - Priority-based
  - Preemptive
  - No time slicing

P1-B

- Memory Management
  - Fixed-size memory pool, no virtual memory
  - A blocking memory allocator
  - Ownership of memory block

P1-B

common.h

```c
#define FALSE              0
#define NULL               0
#define RTX_ERR            -1
#define RTX_OK             0
#define NUM_TEST_PROCS     6

/* Process IDs */
#define PID_NULL           0
#define PID_P1             1
#define PID_P2             2
#define PID_P3             3
#define PID_P4             4
#define PID_P5             5
#define PID_P6             6

/* Process Priority. The bigger t
#define HIGH               0
#define MEDIUM             1
#define LOW                2
#define LOWEST             3
#define PRI_NULL           4

/* Memory Blocks Configuration *
#define MEM_BLK_SIZE       128
#define MEM_NUM_BLKS       32
```

Copyright by Yiqing Irene Huang 2022. All rights reserved

P1 Requirements : User API

- Memory Management: a memory pool which has fixed size of memory block and fixed number of memory blocks.

```
void *request_memory_block()
int  release_memory_block(void *memory_block)
```

P1-A

P1B Adding blocking and ownership

- Processor Management

```
int release_processor()
```

- Process Priority Management

```
int set_process_priority(int process_id, int priority)
int get_process_priority(int process_id )
```

P1-B

Copyright by Yiqing Irene Huang 2022. All rights reserved

P1 Requirements: Processes

- Null Process
  - A system process which does nothing in an infinite loop. PID=0, PRIO=4.
- Test Processes
  - Up to six test processes with PIDs = 1,2, …, 6
  - User level processes, only calls the user APIs
- Initialization
  - Memory, system processes and user processes

All processes never terminate!
No new process created on the fly.

Copyright by Yiqing Irene Huang 2022. All rights reserved

RTX Initialization: Processes

- How does the RTX know which process(es) to create ?

- Pre-defined initialization table
  - An array of records
  - Each record contains spec of a process

Process ID
Initial priority
Initial SP
Initial PC (i.e. entry point)

Copyright by Yiqing Irene Huang 2022. All rights reserved

Initialization Table

- Context_Switching/src/common.h

```c
/* initialization table record */
typedef struct proc_init {
    int m_pid;                /* process id */
    int m_priority;           /* initial priority */
    int m_stack_size;         /* stack size in bytes */
    void (*mpf_start_pc) (); /* entry point of the process */
} PROC_INIT;
```

- The table is an array of these records

Copyright by Yiqing Irene Huang 2022. All rights reserved

Initialization Table Code

```
/* ae_proc.h */
void proc1(void);
void proc2(void);

/* ae_proc.c */
void set_test_procs(PROC_INIT *procs, int num) {
    for(int i = 0; i < num; i++ ) {
        procs[i].m_pid        = (U32)(i+1);
        procs[i].m_priority   = LOWEST;
        procs[i].m_stack_size = USR_SZ_STACK;
     }

    procs[0].mpf_start_pc = &proc1;
    procs[1].mpf_start_pc = &proc2;
    /* other proc setting code not shown */
}
```

Student:

- ae_proc[1-99].c

Lab staff:

- P1 ae_proc[100-199].c
- P2 ae_proc[200-299].c
- P3 ae_proc[300-399].c

Copyright by Yiqing Irene Huang 2022. All rights reserved

RTX Initialization

```
/* rtx.h */

void rtx_init(PROC_INT *proc_info, int num);
```

- RTX will execute one of the processes on success
- It does not return

Copyright by Yiqing Irene Huang 2022. All rights reserved

Processor Virtualization

Crux: How to represent a stream of execution?

- Execution state
  - Machine state (CPU registers)
  - Stack (requires 8-byte alignment)
- Management state (frequently changes)
  - Process state (ready, blocked, et. al.)
- Management information (changes less often)
  - Process ID
  - Priority
  - Entry point

Copyright by Yiqing Irene Huang 2022. All rights reserved

Process Control Block

- Needed for each process
- Describes status and context of a process

```
/* k_inc.h */
/* You need to add more states */
typedef enum {NEW = 0, RDY, RUN} PROC_STATE_E;
```

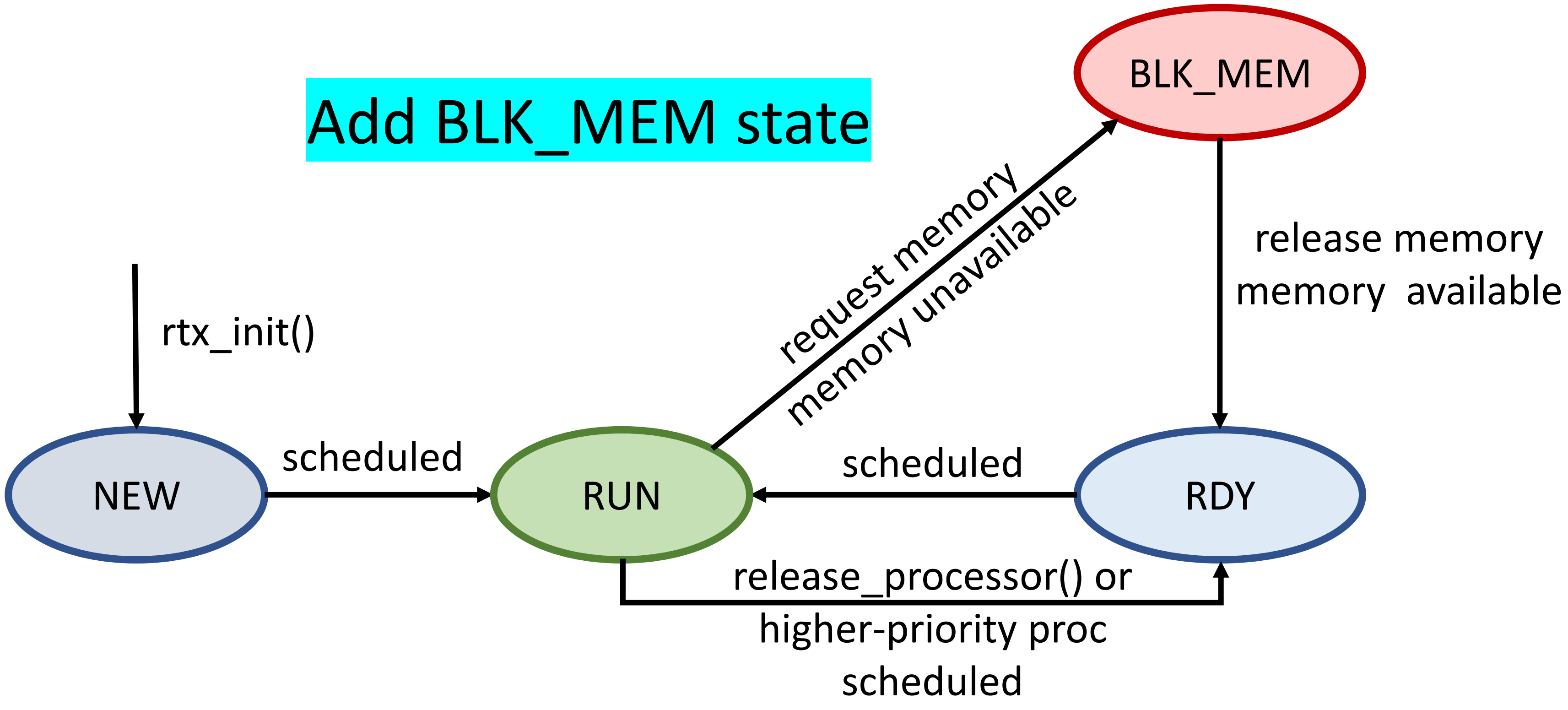
```
typedef struct pcb {
    struct pcb *mp_next;  /* next pcb            */
    U32 *mp_sp;           /* stack pointer       */
    U32 m_pid;            /* process id          */
    U32 m_priority;       /* U32 m_priority      */
    PROC_STATE_E m_state; /* state of the process */
    /* add your own PCB fields here */
} PCB;
```

Copyright by Yiqing Irene Huang 2022. All rights reserved

P1 Process State Transition

```
/* k_inc.h starter code */
typedef enum {NEW = 0, RDY, RUN} PROC_STATE_E;
```

Add BLK_MEM state



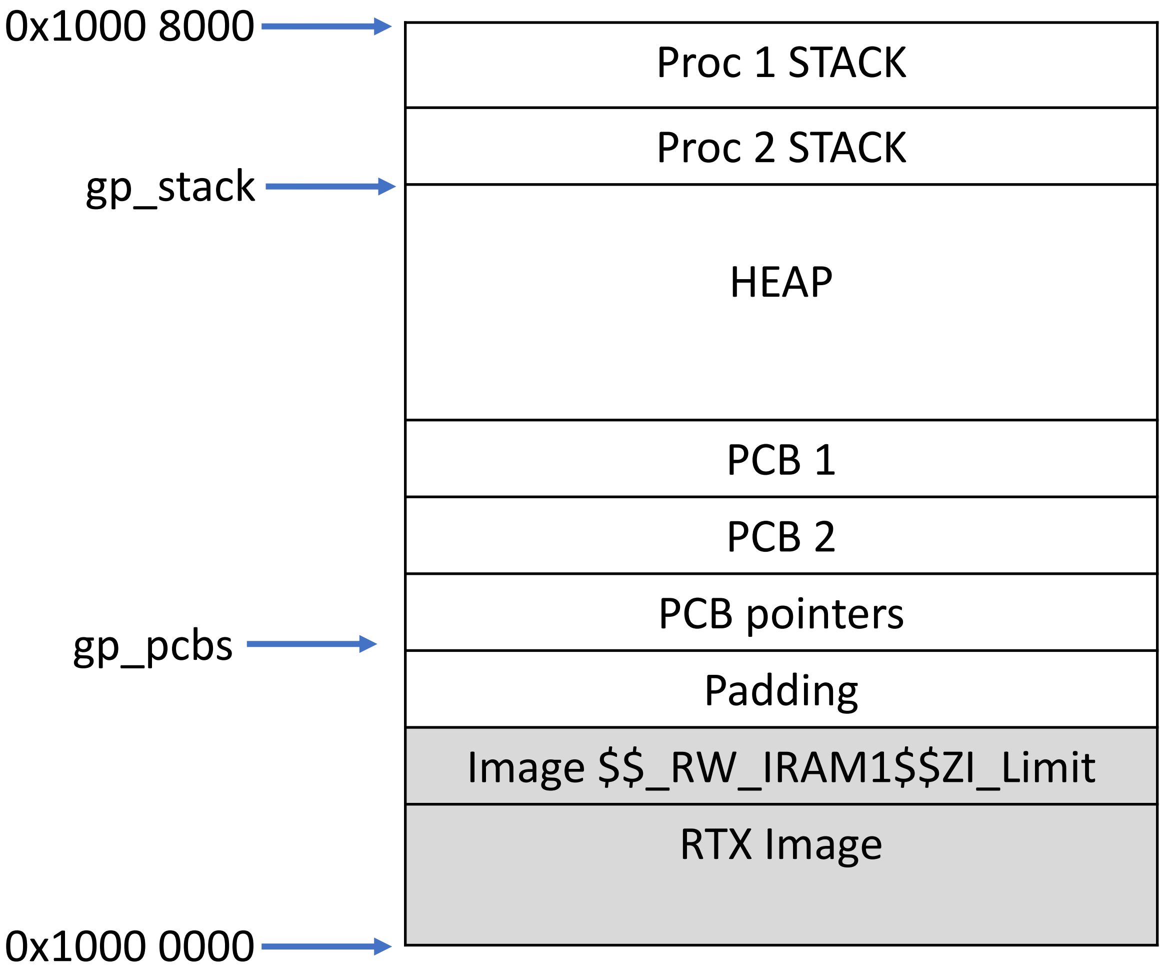
Copyright by Yiqing Irene Huang 2022. All rights reserved

RTX Initialization

- What operations need to be carried out at start-up?
- Initialize all hardware, incl.
  - Board system Initialization
  - Interrupts (~~hardware~~ and software: vector table & traps)
  - Serial port(s) ~~and timer(s)~~
- Create all kernel data structures
  - Memory management kernel data structure
  - Process-control kernel data structure: PCB, kernel stacks
- Create PCBs of all processes
  - allocate stacks
  - privilege level setting using CONTROL register
  - Exception stack frame creation for new processes

Copyright by Yiqing Irene Huang 2022. All rights reserved

IRAM1 Memory Map

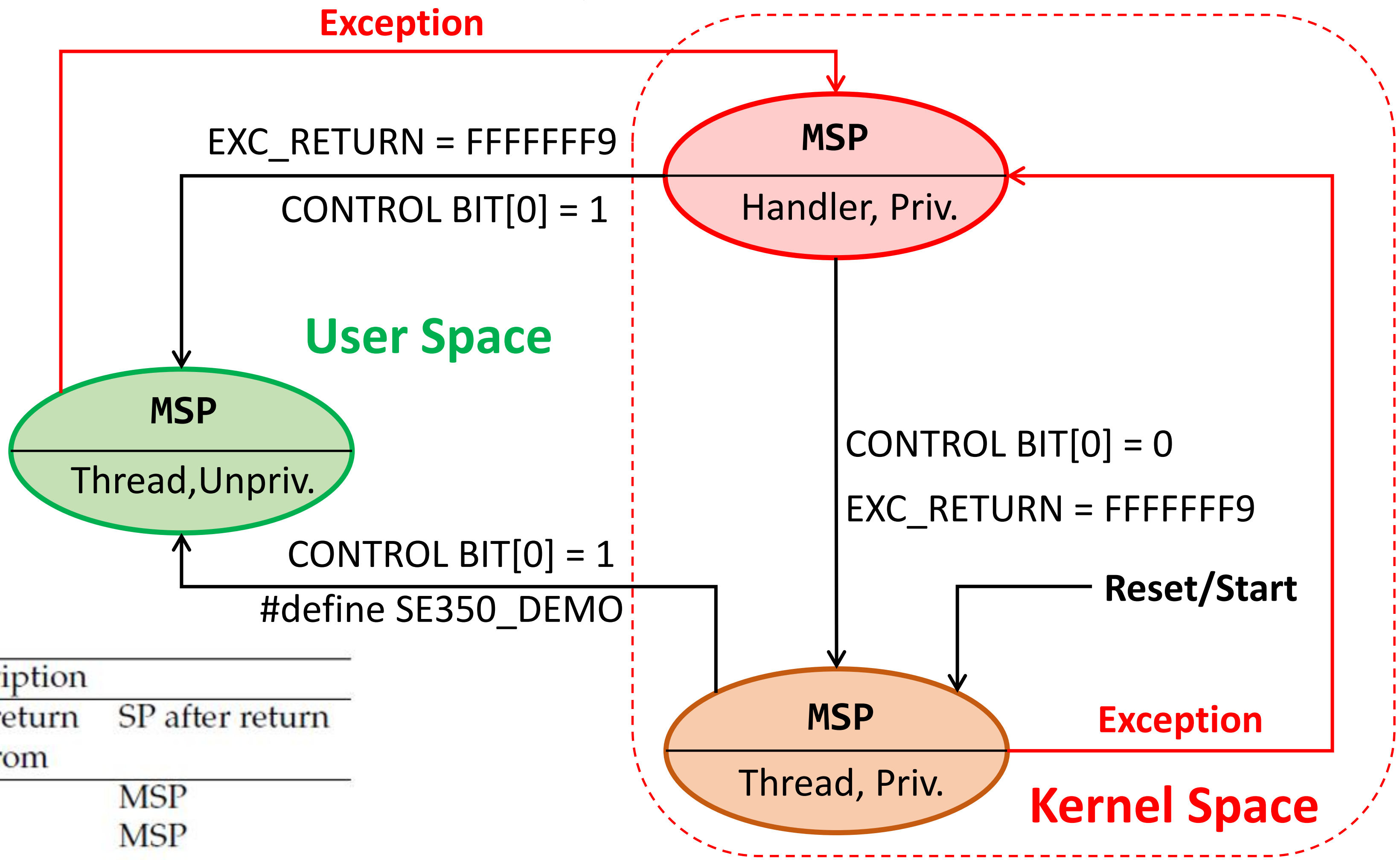


Copyright by Yiqing Irene Huang 2022. All rights reserved

Starter Code Modes and Stacks Diagram

CONTROL

31:3	2	1	0
		SPSEL	nPRIV



EXEC_RETURN

Value	Return Mode	Exception return gets state from	SP after return
0xFFFFFFF1	Handler	MSP	MSP
0xFFFFFFF9	Thread	MSP	MSP
0xFFFFFFFD	Thread	PSP	PSP

Copyright by Yiqing Irene Huang 2022. All rights reserved

Create a New Process

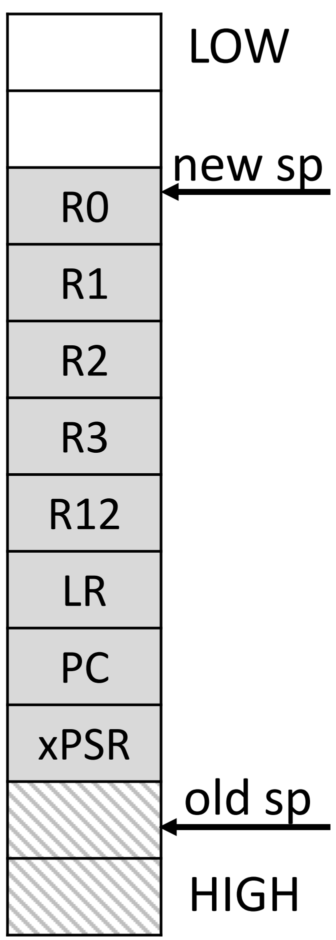
- Manually create the exception stackframe
- Pop off the exception stack frame

```
/* pop off exception stack frame from the stack */
__asm void __rte(void)
{
  PRESERVE8                        ;8B alignment of the stack
  MVN  LR, #:NOT:0xFFFFFFF9        ;set EXC_RETURN, Thread mode, MSP
  CPSIE I                          ;enable interrupts
  BX   LR                          ;load EXC_RETURN to PC
}
```

Copyright by Yiqing Irene Huang 2022. All rights reserved

Exception Stack Frame

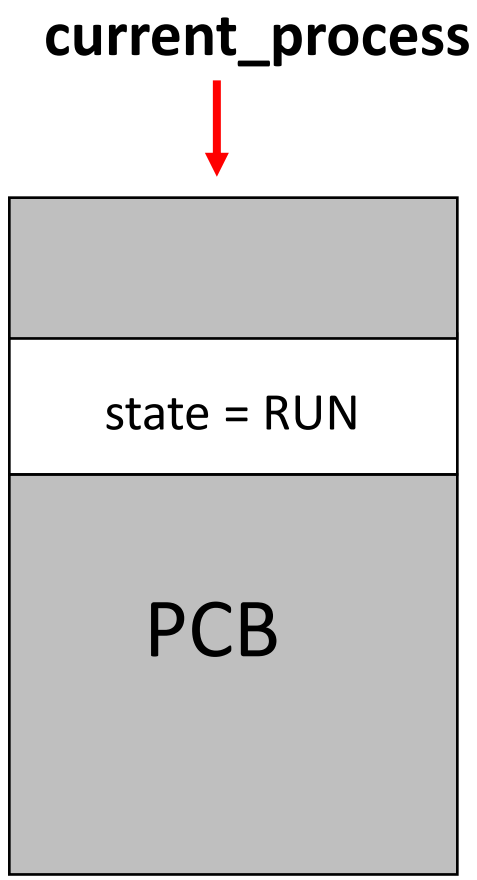
```c
/* k_process.c: process_init()*/

/* stacks grows down, so get the high addr. */
sp = alloc_stack(g_proc_table[i].m_stack_size);
/* process initial xPSR  */
*(--sp)  = INITIAL_xPSR;
/* PC contains the entry point of the process */
*(--sp)  = (U32) ((g_proc_table[i]).mpf_start_pc);
for (int j = 0; j < 6; j++ ) { /*R0-R3, R12, LR */
    *(--sp) = 0x0;
}
(gp_pcbs[i])->mp_sp = sp;
```



Copyright by Yiqing Irene Huang 2022. All rights reserved

The Current Process

- The `current_process` variable:
  - OS must know, which process currently executes.
  - It always refers to PCB of currently executing process.
  - Only works for single-core processor



Copyright by Yiqing Irene Huang 2022. All rights reserved

Process Switch

- Policy
  - Scheduler selects the next process to execute
- Mechanism - Context switch to the new process
  - update state of PCB
  - update ready queue
  - switch the stacks of the processes

Copyright by Yiqing Irene Huang 2022. All rights reserved

Scheduling Requirements

- No time slicing
- Fixed, priority-based scheduling
- Preemption
- Each process has assigned priority
  - Highest priority process executes first
  - First come, First served for processes of same priority

```
/* common.h, included by rtx.h */
#define HIGH      0
#define MEDIUM    1
#define LOW       2
#define LOWEST    3
#define PRIO_NULL 4 /* hidden */
```

Copyright by Yiqing Irene Huang 2022. All rights reserved

Preemption

- A higher priority process becomes ready
  - The kernel will run this higher priority process
  - The current running process is preempted
    - Change to READY
    - Remains its position in the ready queue
- This should never happen!!!
  - P1 is in READY and P2 is in RUN and
  - Priority (P1) > Priority(P2)

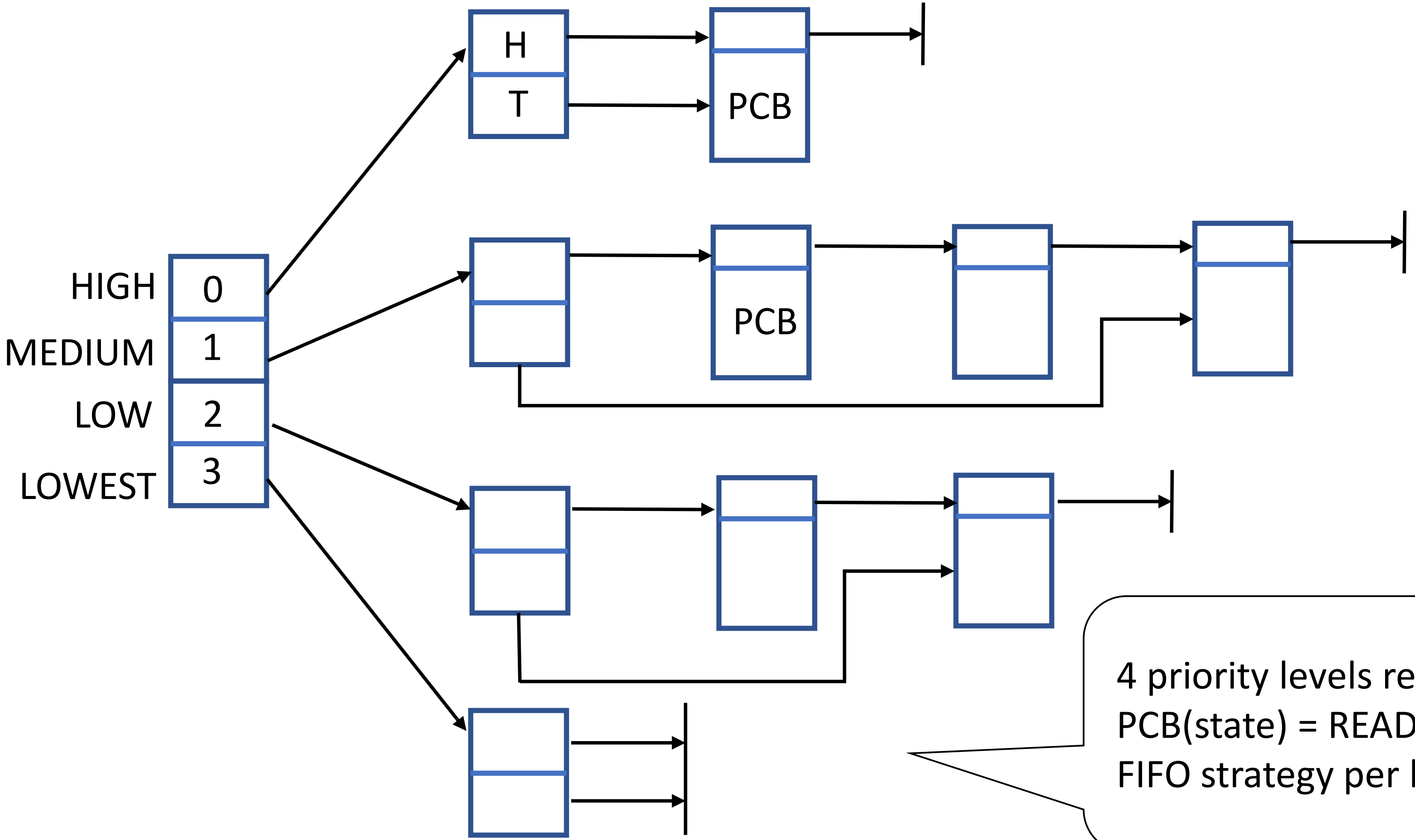
Copyright by Yiqing Irene Huang 2022. All rights reserved

Scheduling Procedure

- The kernel invokes scheduler
- Scheduler selects highest-priority ready process
- The process_switch(new_proc) makes the selected process to execute

Copyright by Yiqing Irene Huang 2022. All rights reserved

Scheduling: Ready Queues



4 priority levels ready Q
PCB(state) = READY
FIFO strategy per level

Copyright by Yiqing Irene Huang 2022. All rights reserved

Scheduling: Null Process

- CPU must execute something
- What to do when ready queues are empty?
  – Possible solution: NULL process
  – NULL has lowest priority (hidden) and is always ready to run
- Basic example (for non-preemptive kernel)

```
void null_process() {
        while(1) {
                release_processor();
        }
}
```

How to change the code in a preemptive kernel?

Copyright by Yiqing Irene Huang 2022. All rights reserved

Scheduling Examples

- P1 (LOW) is running, P2 (HIGH) becomes ready from a blocked state
  - P2 should run, P1 keeps its position in the LOW ready queue (i.e. gets added to the head of LOW ready queue).
- P1 (LOW) is running, P3(LOWEST) becomes ready from a blocked state
  - P1 continue to run, P3 gets added to the back of the LOWEST ready queue.
- P1 (LOW) is running, P4(LOW) is the head of the LOW ready queue, P1 calls release_processor()
  - P1 becomes ready, moves to the back of the LOW ready queue. P4 starts to run
- P1(HIGH) is running and changes its priority to HIGH, P1 should continue to run even P2(HIGH) is ready.

Copyright by Yiqing Irene Huang 2022. All rights reserved

Context Switching

1. Save context of currently executing process
2. Change the process's state back to READY
3. Update `current_process` to new process
4. Set state of new process to RUN
5. Restore context of `current_process`
6. Execute `current_process` by switching the stacks

k_process.c: process_switch()

Copyright by Yiqing Irene Huang 2022. All rights reserved

Memory Allocation

- A **blocking** memory allocator

```
void * k_request_memory_block() {
        atomic(on)
        ptr = k_request_memory_block_nb();
        while (ptr is null) {
                put the PCB on blocked_memory_q
                set the PCB state to BLOCKED_ON_RESOURCE
                k_release_processor()
                update ptr accordingly
        }
        atomic(off)
        return ptr;
}
```

Copyright by Yiqing Irene Huang 2022. All rights reserved

Memory Deallocator

- Note this may cause preemption to happen.

```
int *k_release_memory_block(void *mem_blk) {
        atomic(on)
        if (mem_blk is invalid)
                return ERROR_CODE
        if (blocked on memory resource q is empty) {
                put the mem_blk to free_memory queue/list
        else
                dequeue a blocked-on-memory PCB
                handle_process_ready(PCB)
                assign the mem_blk to the PCB
                make scheduling decision
                do process switch if there is a need
        }
        atomic(off)
        return SUCCESS_CODE
}
```

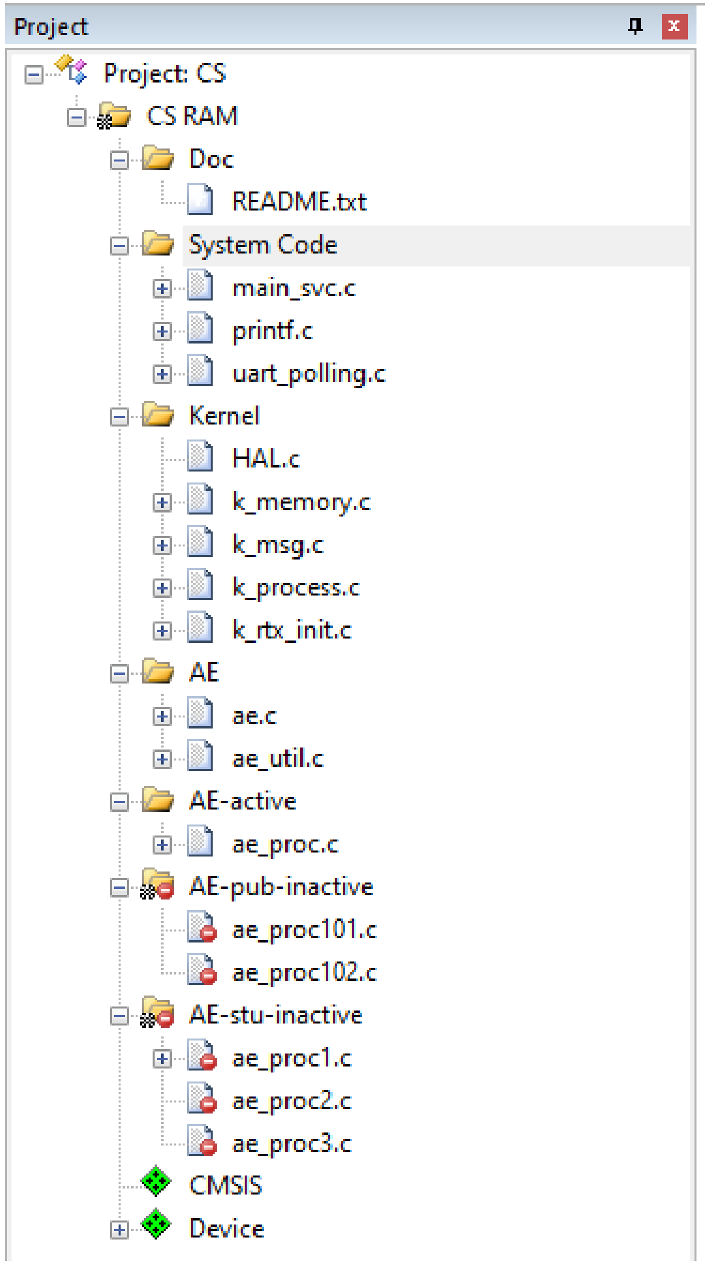
Copyright by Yiqing Irene Huang 2022. All rights reserved

The Testing Framework

Copyright by Yiqing Irene Huang 2022. All rights reserved

Project Structure

- System code
- Kernel
  - Kernel objects and services
- AE*
  - AE
    - AE_ENABLE macro
  - AE-active:
    - The active testing suite .c file
  - AE*-inactive:
    - Testing suites
    - One .c is per testing suite



Copyright by Yiqing Irene Huang 2022. All rights reserved

Testing Framework Requirements

- User files header files:
  - Do not change existing function prototypes. You may change their implementations
- Kernel header files:
  - Do not change prototype of functions which appear in rtx.h and rtx_ext.h files
- You may add new functions to .c files and their declarations to the corresponding modifiable .h files.
- Keep existing project source groups in IDE
- You may add new source groups in IDE
- Do not modify existing project file system structures
- You may add new files/folders into src/ directories

Existing Function Prototype Cannot be Modified!

Files	Adding new stuff
rtx.h	NO
common.h	NO
ae.h	NO
ae_util.h	NO
ae_proc.h	YES
rtx_ext.h	YES
common_ext.h	YES

Copyright by Yiqing Irene Huang 2022. All rights reserved

Testing

- Minimum one Test Suite
- Minimum three test cases
- Context_Switching/uart1.log has the simulator output of UARTs
- Submit expected output files
- We also write our own test suites

```
├── CS.uvoptx
├── CS.uvprojx
├── DebugConfig
├── doc
├── EventRecorderStub.scvd
├── expected_output
│   ├── G99-TS1.log
│   ├── G99-TS2.log
│   └── G99-TS3.log
├── Listings
├── Objects
├── RAM.ini
├── RTE
├── SIM.ini
├── src
```

Copyright by Yiqing Irene Huang 2022. All rights reserved

Git Submission

Copyright by Yiqing Irene Huang 2022. All rights reserved

Frequently Used Git Commands

- P1 submission commit should be tagged with “p1-submit”

General Git Commands	Git Tag Commands
git clone <url>	git tag –a p1-submit –m “G99 p1 submission tag”
git pull	git push origin p1-submit
git add	git push origin --tags
git commit –m “commit message”	git tag
git push	git show lab3-submit
git status	git log –pretty=oneline
git diff	git checkout lab3-submit
https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository	git tag –d <tagname>
https://git-scm.com/book/en/v2/Git-Basics-Tagging	git push origin –delete <tagname>

Copyright by Yiqing Irene Huang 2022. All rights reserved

Project Submission

- Commit your changes

```
git commit –m "commit message"
```

- Push the commit to remote server

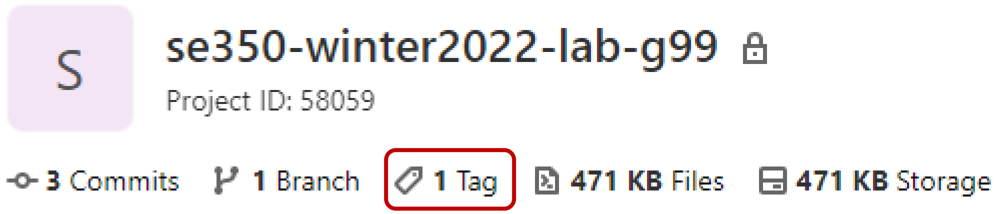
```
git push
```

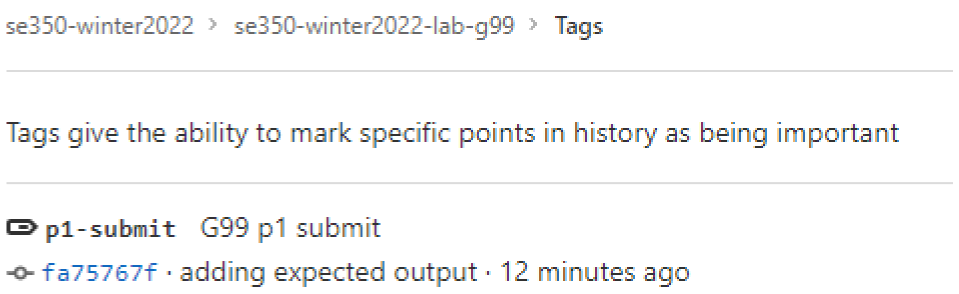
- Tag the commit you want to submit with "p1-submit"

```
git tag -a p1-submit –m "G99 P1 submit"
```

- Push the local tag to remote server

```
git push origin p1-submit
```





Copyright by Yiqing Irene Huang 2022. All rights reserved

Double check your submission

```
git clone --depth 1 --branch p1-submit <repo_url>
```

Copyright by Yiqing Irene Huang 2022. All rights reserved

References

1. Dasiewicz, Paul, A non-preemptive RTX Design Documentation
2. LPC17xx User’s Manual
3. ARM Compilation Tools Version 5.0 Developer Guide
4. Software Interface Standard for Arm Cortex-based Microcontrollers, CMSIS Version 5.7.0

Copyright by Yiqing Irene Huang 2022. All rights reserved

Acknowledgement

Slides modified from

- P. Dasiewicz
- J.C. Petkovich

Copyright by Yiqing Irene Huang 2022. All rights reserved

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Thank you!

Electrical and Computer Engineering Department

Copyright by Yiqing Irene Huang 2022. All rights reserved