

# EE445M/EE380L.12

## Embedded and Real-Time Systems/ Real-Time Operating Systems

### Lecture 8: Memory Management, Heap, Processes, Process Management

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

1

## Operating System

- Manage computer system resources
  - CPU, processors
    - Threads
  - Storage, flash/disc
    - Files
  - Memory, RAM
    - Heap, processes

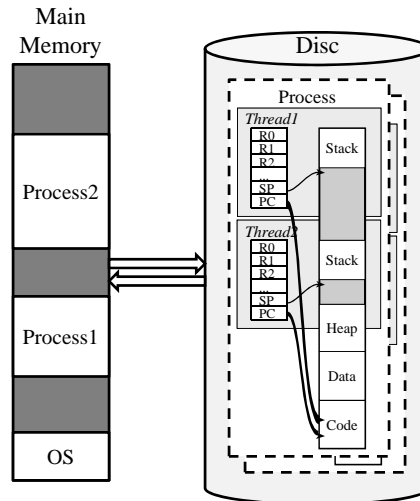
Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

2

## Memory Management

- Sharing
  - Per-thread: stack
  - Per-program/-process: heap, code, data
- Allocation
  - Static, permanent
    - Globals, OS code
  - Dynamic, temporary
    - Stack, heap, process swapping
- Protection
  - Access control



Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

3

## Fragmentation

- Internal
  - Wasted space inside allocated region
  - Convenience of the operating system
  - Contains no information
  - Wasted in order to improve speed or provide for a simpler implementation
- External
  - Unusable storage is outside the allocated regions
  - Largest block that can be allocated is less than the total amount of free space
  - Occurs because memory is allocated in contiguous blocks
  - Occurs over time as free storage becomes divided into many small pieces
  - Worse when application/OS allocates/deallocates blocks of storage of varying sizes

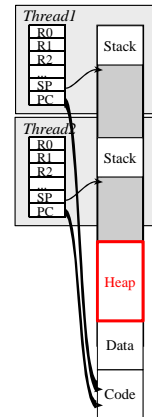
Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

4

# Heap

- Separate piece of main memory
  - “Memory region” in  $\mu$ COS-II
- Managed by the operating system
  - Initialization **Heap\_Init** called by OS during the initialization phase
- Used for temporary allocation
  - Allocation **Heap\_Malloc** called by user or OS
  - Deallocation **Heap\_Free** called by user or OS

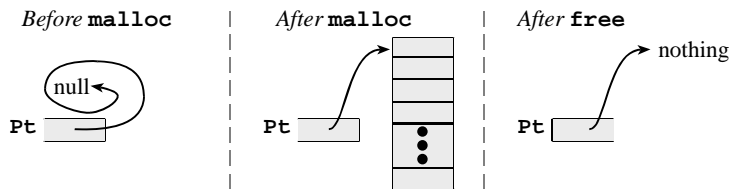


Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

5

# Dynamic Memory Allocation



```
void Function(void){
    int i;
int *pt; int pt[20];
    // allocate 20 words
pt = (*int)Heap_Malloc(4*20);
    for(i = 0; i < 20; i++)
        // put data into array
        pt[i] = i;
Heap_Free(pt);
}
```

```
int *Pt;
void Begin(void){
    // allocate 20 words
    Pt = (*int)Heap_Malloc(4*20);
}
void Use(void){ int32_t i;
    for(i = 0; i < 20; i++)
        // put data into array
        Pt[i] = i;
}
void End(void){
    Heap_Free(Pt);
}
```

Lecture 8

## Heap Manager

- Heap\_Init
  - Allocate & initialize heap memory
    - Statically allocated storage assigned by compiler
 

```
static long Heap[500]; // 2000 byte heap
```
- Heap\_Malloc
  - Allocate block in heap free space
    - Must use contiguous allocation
    - First fit, best fit, worst fit
- Heap\_Free
  - Reclaim block into heap free space

Heap\_4C123.zip

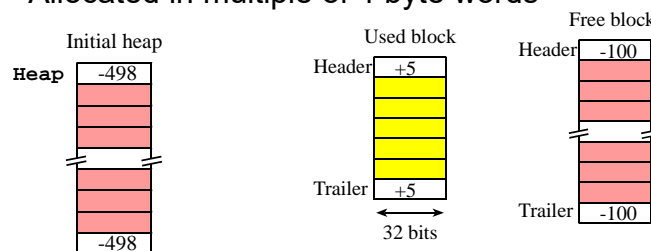
Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

7

## Heap Manager Example

- Blocks of variable size
  - Size counter at beginning/end of each block
    - Positive if used (allocated), negative if free
  - Internal fragmentation
    - Overhead for size header/trailer
    - Allocated in multiple of 4 byte words



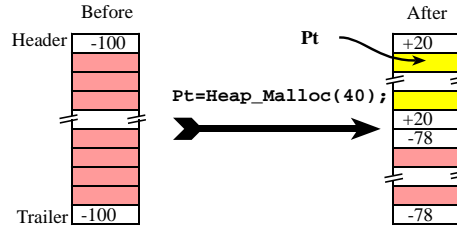
Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

8

## Heap\_Alloc

- Allocate block
  - Find a free block
    - Uses first fit
  - Free block is divided into two parts
    - New free block is smaller
  - A pointer to the allocated block is returned
  - Block may not be large enough to split
    - Allocate the big block, internal fragmentation



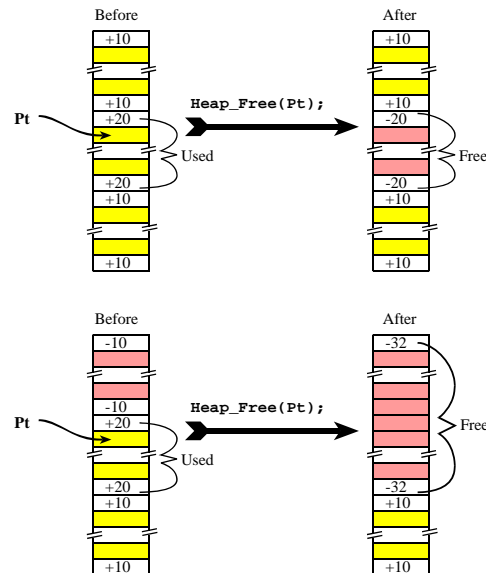
Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

9

## Heap\_Free

- Four cases
  - No merge
  - Merge above
  - Merge below
  - Merge both above and below



Lecture 8

## Knuth's Buddy Allocation

- Maintain heap as collection of blocks each with a size of  $2^m$
- When user requests a block of size  $n$ 
  - Find smallest block with  $2^m \geq n$
  - Split block into half until best fit
- When user releases a block
  - Merge with other half (buddy block of same order), if possible

Lecture 8

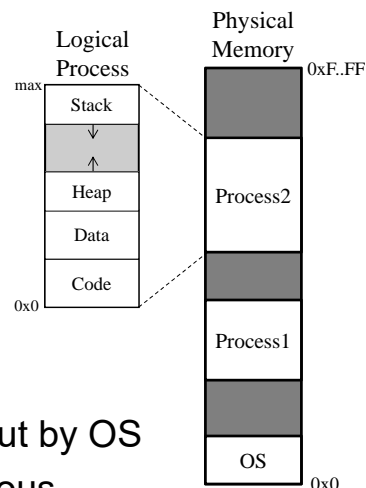
J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

Final Exam 2010

11

## Processes

- OS manages processes
  - CPU scheduling
  - Code/data memory
- Independent programs
  - Separately compiled
  - Logical address space
- Brought in/out of memory
  - On load/exit, swapped in/out by OS
  - Contiguous or non-contiguous



Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

12

# ELF Files

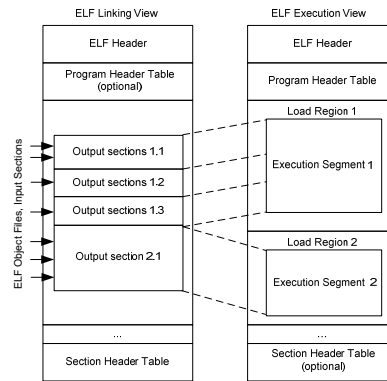
- Executable and Linkable Format (ELF)

- Linking: sections

- Object files -> executables
- Code (RO / .text)
- Data (RW / .data)
- Zero data (ZI / .bss)
- String/symbol table
- ... (debug info) ...

- Execution: segments

- Executable process image
- Contiguous load regions
- One or more sections per segment



Source: infocenter.arm.com

```
C:\Keil_v5\ARM\ARMCC\BIN\fromelf.exe --text User.axf
```

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

13

# ELF Executable

- Process memory image

- Load regions/segments

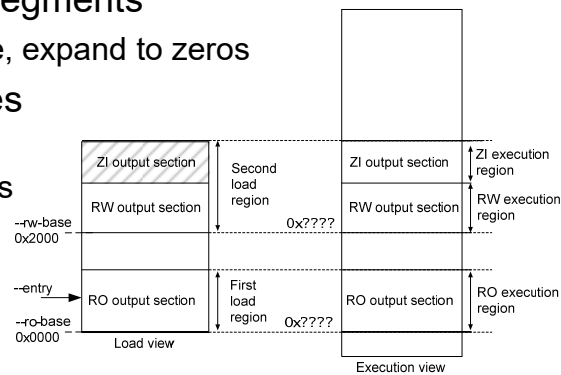
- ZI empty in file, expand to zeros

- Base addresses

- Execution vs. load addresses

- Entry point

- Starting address of execution



Source: infocenter.arm.com

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

14

## Address Translation

- Virtual addresses in process
  - Compiler generated programs on disk
  - Location of & references to code and data
- Physical addresses in main memory
  - Need to map virtual into physical addresses
  - Compile time: generate for known location
  - Load time: relocation by OS, dynamic linking
  - Run time: software or hardware, virtual memory

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

15

## Compile-Time Translation

- Virtual = physical addresses
  - Compiler/linker generate absolute addresses
  - Loaded at fixed, pre-defined location
  - Swap processes if overlapping
- Multi-programming
  - Multiple processes in memory at same time
  - Compile for non-overlapping locations?
  - Swap overlays on every context switch?

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

16



## Run-Time Position Independence

- Position-independent code (PIC)
  - Code/RO segment compiled to run anywhere
  - All references within segment are PC-relative
    - Default for ARM short jumps: **B**, **BL**, **Bnn** (not: **BX**)
    - Data within segment: **LDR Rx,=v / [PC,#n]**
- Position-independent data
  - References from code to data/RW segment
  - R9 as static base (SB) register
    - Must point to base address of data/RW segment
    - All references as offsets added to R9/SB

```

...
LDR r1,[r9,#ofs]
...
LDR r0,=ofs
ADD r0,r9,r0
LDR r0,[r0]
...
    
```

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

17

## Load-Time Relocation

- Relocatable process image
  - Compiler/linker place code/data in segments
    - ELF symbol table
  - Generate dummy addresses for references
    - ELF relocation table entries
  - Patch addresses with real location on load

```

...
dummy                               R_ARM_THM_CALL → dummy
EBFFFFFFE BL dummy ; #ofs = -4 (f) EB000000 BL f ; #ofs = 0
E59F00nn LDR r0,[pc,#n] ; [addr_d] E59F00nn LDR r0,[pc,#n] ; LDR r0,=d
E5900000 LDR r0,[r0] E5900000 LDR r0,[r0]
...
addr_d                               R_ARM_THM_ABS32 → addr_d
00000000 DCD 0x00000000 dddddddd DCD 0xdddddddd
    
```

C:\Keil\_v5\ARM\ARMCC\BIN\fromelf.exe -y -r User.axf

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

18

## Dynamic Linking

- Resolve references to external symbols
  - Code / data shared between processes
  - OS kernel and shared libraries
- ELF dynamic linking segment (.dynamic)
  - Dynamically linked external symbol table
    - Addresses must be provided by loader
  - Standard relocation entries

C:\Keil\_v5\ARM\ARMCC\BIN\fromelf.exe -y -r User.axf

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

19

## OS Kernel Calls

- Static or dynamic linking
  - Static linking to fixed location at compile time
  - Dynamic linking using relocation at load time
- Supervisor Calls
  - Trigger SVC exception from user code
  - SVC handler in kernel

```
EXTERN ST7735_Msg [DYNAMIC]
```

```
; Long call RAM->ROM
Display_Msg
  LDR R12,=ST7735_Msg
  BX  R12
```

```
OS_sleep
  SVC #2
  BX  LR
```

```
OS_Time
  SVC #3
  BX  LR
```

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

20

# SVC Handler

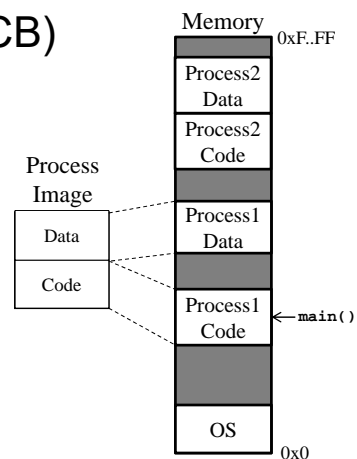
- Exception stack:
- R0-R3, R12
  - LR
  - Return address
  - PSR

```

SVC_Handler
    LDR R12,[SP,#24]    ; Return address
    LDRH R12,[R12,#-2] ; SVC instruction is 2 bytes
    BIC R12,#0xFF00    ; Extract ID in R12
    LDM SP,{R0-R3}     ; Get any parameters
    ...
    BL OS_xxx          ; Call OS routine by ID
    ...
    STR R0,[SP]        ; Store return value
    BX LR              ; Return from exception
    
```

# Process Management

- Process Control Block (PCB)
  - Process ID (PID)
  - Code & data segment
  - One or more threads
    - Main and child threads
  - Priority
  - ...
- Parent process in TCBs



## Process Creation

- Unix
  - **fork()**
    - Create copy of current process
  - **exec()**
    - Replace current process with image on disk
  - **init** process (process ID, PID = 0/1)
    - Mother of all processes created by OS
- Windows
  - **CreateProcess()**
    - Create new process and load program image

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

23

## Process Termination

- Unix
  - **exit()**
    - Terminate current process
    - OS frees all resources (memory, thread, ...)
    - Returns exit status
    - Automatically invoked on return from **main()**
- Windows
  - **ExitProcess()**
    - Likewise

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

24

## Lab 5

- User program (**RTOS\_Lab5\_User**)
  - Position-independent code & data (requires full Keil license\*)
  - Dynamic linking for display driver calls (**ST7735\_xxx**)
  - SVC traps for **OS\_xxx** calls (incl. **OS\_AddThread**)
- OS (**RTOS\_Lab5\_ProcessLoader**)
  - Heap manager → **develop in this lab**
    - Dynamic allocation of process memory
  - FAT file system SDCFile\_4C123.zip
    - Read user programs compiled on PC
  - ELF file loader <https://github.com/gerstl/elfloader>
    - Allocate, load from SD, link/relocate, call **OS\_AddProcess**
  - Process management → **develop in this lab**
    - Process creation: **OS\_AddProcess** (with 1 initial thread)
    - Process termination: when last thread is killed
    - SVC handler & static base (SB) register (R9)

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

\* Email Prof or TAs

25

## ELF Loader

- Configuration (**loader\_config.h**)

```

#define VALVANOWARE // for this class

#ifdef VALVANOWARE
#include "ff.h"
#include "heap.h"
#include "os.h"

#define LOADER_OPEN(fd,path)          f_open(fd, path, FA_READ)
#define LOADER_READ(fd,buf,size)     f_read(fd, buf, size)
#define LOADER_CLOSE(fd)             f_close(fd)
#define LOADER_ALLOC(size)           Heap_Alloc(size)
#define LOADER_JUMP_TO(entry,code,data) OS_AddProcess(entry, code, data)
...

```

- Basic operation (**loader.c/.h + elf.h**)

```

int exec_elf(const char *path, const ELFEnv_t *env) {
    LOADER_OPEN(&f, path); // open & read ELF header
    ...
    text = LOADER_ALLOC(<code_size>); // allocate & load code segment
    LOADER_READ(f, text, <code_size>);
    ...
    data = LOADER_ALLOC(<data_size>); // allocate & load data segment
    LOADER_READ(f, data, <data_size>);
    ... // relocation using 'env'
    LOADER_CLOSE(f);
    return LOADER_JUMP_TO(entry, text, data); // add OS process
}

```

## Calling ELF Loader

- Provide symbol table for relocation
  - Mapping symbol names to OS addresses
  - Used to patch binary on loading

```
static const ELFSymbol_t syntab[] = {
    { "ST7735_Message", ST7735_Message }
};

void Interpreter() {
    ELFEnv_t env = { syntab, 1 };
    ...
    if (!exec_elf(<filename>, &env)) { ... }
    ...
}
```

Lecture 8

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

27