Operating System (OS)
CS232

Memory Virtualization: Memory API

Dr. Muhammad Mobeen Movania

Outlines

- Memory Management in Unix/C Program
- Two Types of Memories
  - Stack (static/compile time)
  - Heap (dynamic/run time)
- Common errors
- System calls
- More memory allocation functions
- Summary

Memory Management in Unix/ C Program

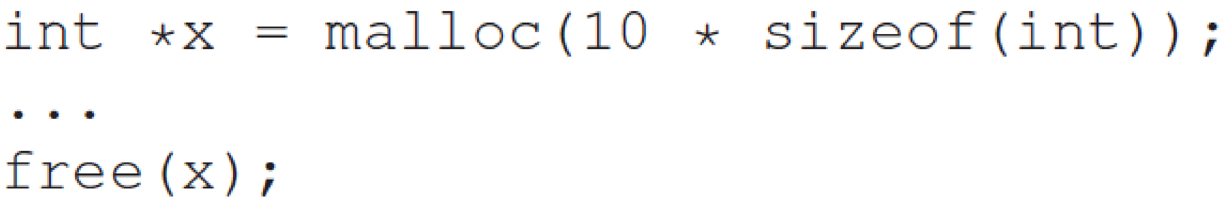
- Two types of memory allocations
  - Stack
  - Heap
- Stack memory allocations
  - Automatic management (by compiler)
  - Stores function parameters, local variables, return addresses, etc.
  - Memory gets freed once function scope ends

Code Example

```
#include <stdio.h>
int main(int argc, char* argv[])
{
    int num = 10; //num’s life start here
    printf(“Num: %d\n”, num);
    return 0;
}//num gets deleted automatically
when function scope ends
```

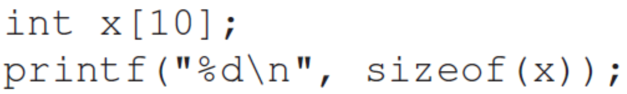
Memory Management in Unix/ C Program

- Heap
  - Explicitly handled by the programmer
  - Memory must be reclaimed by the programmer using the free function

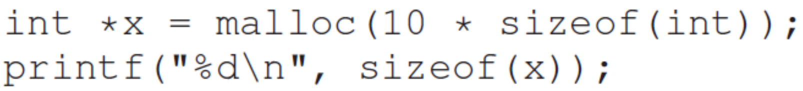


```
int *x = malloc(10 * sizeof(int));
...
free(x);
```

  - Use sizeof to calculate the total number of bytes to allocate
  - But beware:
  - sizeof(pointer) != sizeof(static array)



```
int x[10];
printf("%d\n", sizeof(x));
```



```
int *x = malloc(10 * sizeof(int));
printf("%d\n", sizeof(x));
```

Common Errors in Memory Management

- Failure to allocate memory

```
char* src = “hello”;
char* dst; //dst is a null pointer
strcpy(dst, src);
```

- Not allocate enough memory

```
char* src = “hello”;
char* dst = (char*)malloc(strlen(src));
// char* dst = (char*)malloc(strlen(src)+1);
strcpy(dst, src);
```

- Forgetting to initialize or free memory

Common Errors in Memory Management

- Using memory after freeing (dangling pointer)

```
int* i=(int*)malloc(sizeof(int));
free(i);
*i = 100; //Boom: i is already deallocated
```

- Freeing memory repeatedly (double free)

```
int* i=(int*)malloc(sizeof(int));
free(i);
free(i); //Boom: i is already deallocated.
```

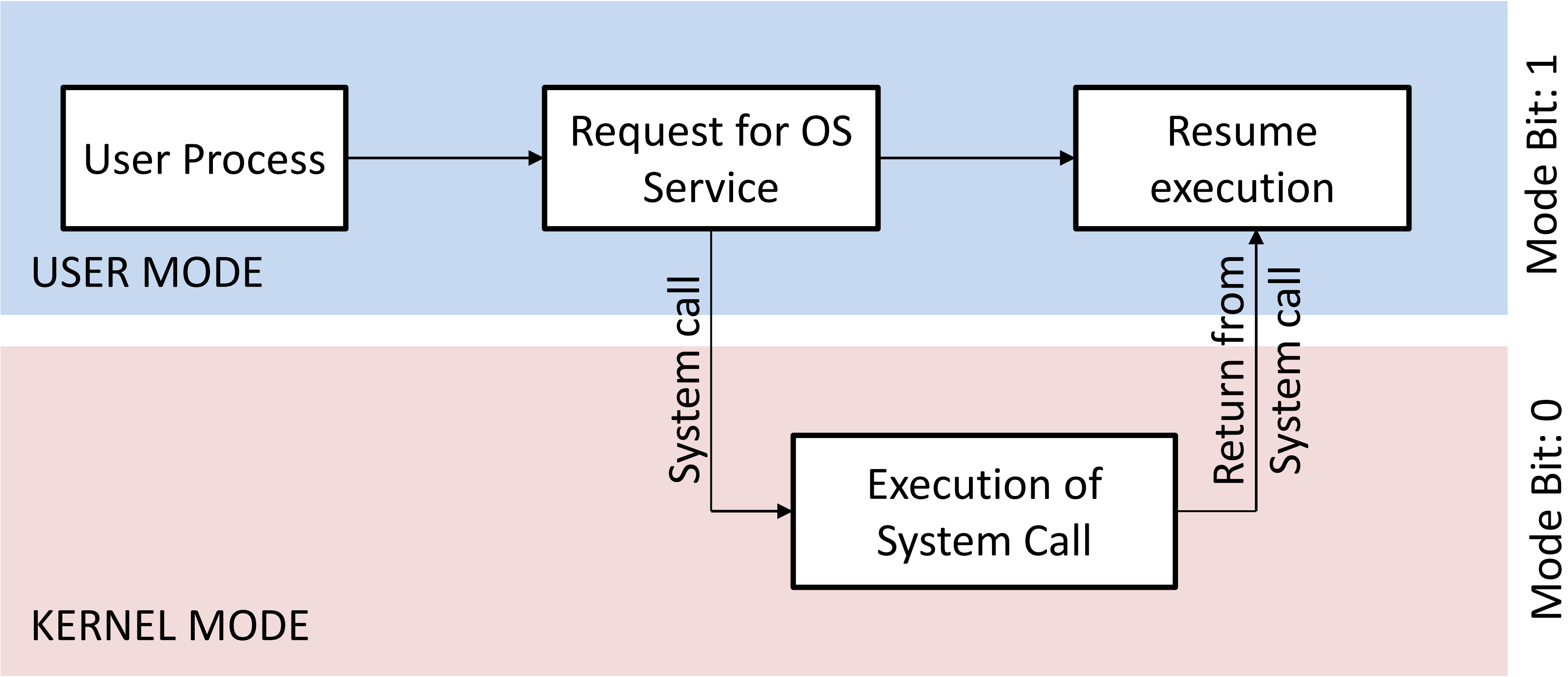
- Calling free with invalid (non malloc’ed) address

```
int i=10;
int* pi = &i;
free(pi)
```

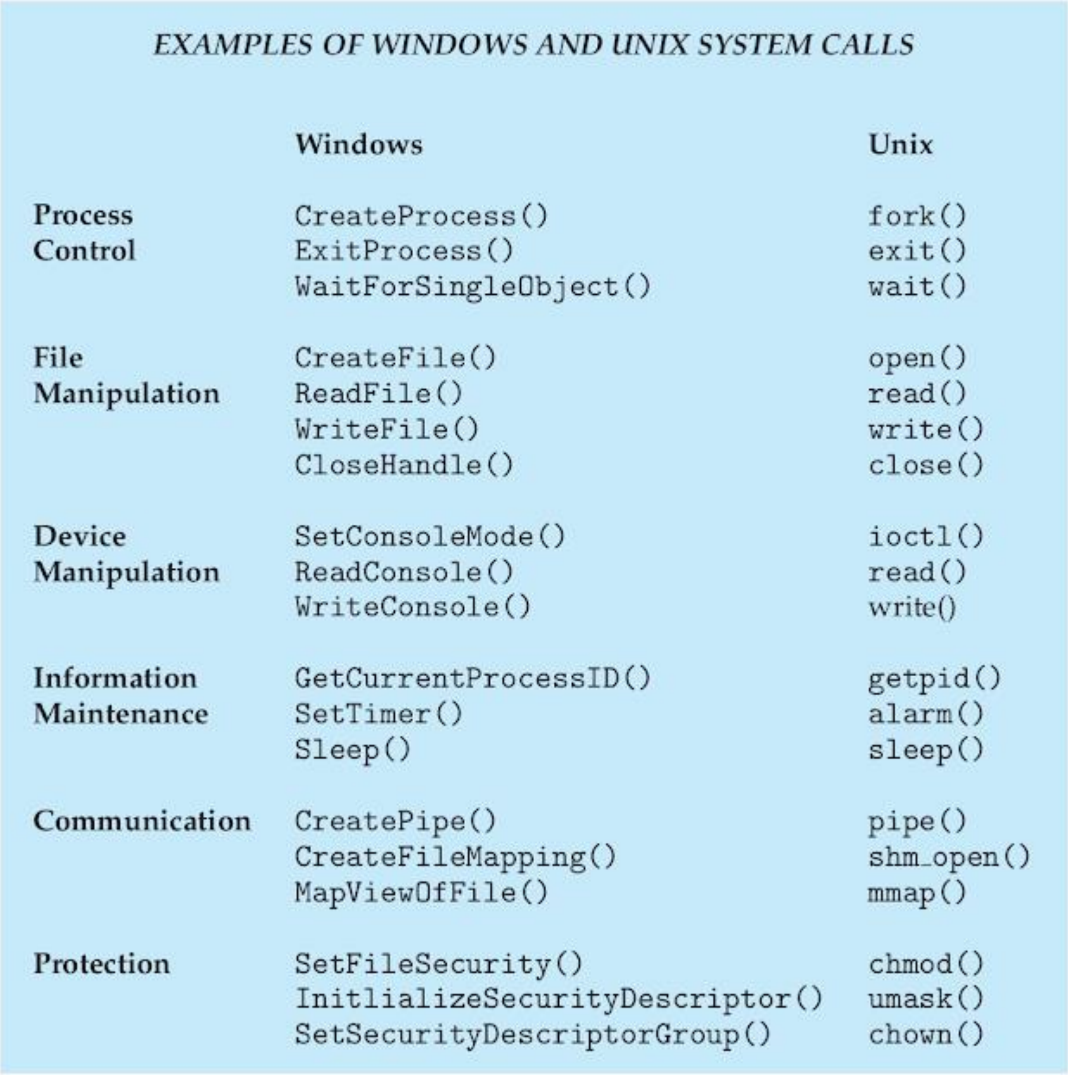
System Calls

- Operating System works in two modes
  - User Mode
  - Kernel Mode
- Special functions that are executed in kernel mode
  - Most of these functions access hardware resources so care has to be taken
- A **system call** is a mechanism that provides the interface between a process and the operating system.
  - Offers the services of the operating system to the user programs via API

System Call Flow



Windows and Linux System Calls



EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shm_open()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

System calls

- malloc() and free() both are **library calls**
- Other library calls include calloc(), realloc(), etc.
- malloc() and free(), etc., use the brk() system call
  - brk, sbrk change the size of the heap
- mmap() is another system call for requesting memory from OS
  - mmap creates an anonymous memory region in your program
  - the region is in swap space not with any file so it may be treated similar to heap

More memory allocation functions

- calloc()
  - allocates memory and also zeroes it before returning
  - prevents some errors where you assume that memory is zeroed and forget to initialize it yourself
- realloc()
  - can also be useful, when you’ve allocated space for something (say, an array), and then need to add some more space to it
  - it makes a new larger region of memory, copies the old region into it, and returns the pointer to the new region

Summary

- We talked about the two types of memory allocations: stack and heap
- We looked at some common errors during memory allocation
- We introduced some API responsible for memory allocation, specifically, malloc, free, calloc, realloc functions
- We saw that the memory allocation functions make system calls to allocate memory dynamically