LECTURE 2

PROFESSOR
Office hours: Friday 2:00
334 WVH

TA: MONDAY 10:00-11:30
166 WVH.

RECAP (last week's class):

1. Memory Maps
2. Programs
3. Simple OS

Simple 16-bit processor
64 KB addresses, similar to MSDOS

FFFE

OS

| INTERFACE |
| STACK |
| HEAP |
| PROGRAM- CODE |

0000

Single user OS
one program at a time
ADDED MULTITASKING

OS  S → Stack
     H → Heap

S ↓ H ↑  P1  → Program code.

Part of the stack has all the information till the termination of the process.

Context switching:

Context switching can be achieved by storing the previous stack pointer of the SP in the main memory and retrieving later.
In reality both the processes exist in the same address memory.

Memory protection prevents one process from writing into the address space of another.

Without mem protection debugging becomes close to impossible.

More about our simple OS:

- 8 general registers
- Program counter
- Stack pointer

Now, we add 2 more registers:

- Bounds
- Base
If the OS can set the value of the above pointers, then any process can set the pointer
↓
can be a big loophole and leads to various security threats like stack overflow attacks

Supervisor privilege comes into picture.

INTERRUPT
INT XX
\[ \rightarrow \text{address} \]
\[ \rightarrow \text{makes a system call, which executes interrupt on that address.} \]

Now there are new complexities in context switching after base & bounds pointers.

Context:
old & Stack pointer
new & base + bounds

before: switch from \( P_1 \) to \( P_2 \).
\( P_1 \). Saved \( SP = SP \).
\[ SP = p_2 \cdot \text{Saved} \cdot sp \]

RET

\[ \text{New:} \]
\[ p_1 \cdot \text{Saved} \cdot sp = sp \]
\[ \text{base} = p_2 \cdot \text{base} \]
\[ \text{bounds} = p_2 \cdot \text{bounds} \]
\[ SP = p_2 \cdot \text{Saved} \cdot sp \]

base and bounds have a fixed values so we need not save the values.

\[ g(c) \quad h(c) \]

return from \( Z \):

\[ \text{Stack} \quad X \]
\[ \text{Y} \quad \text{Z} \]

return from \( Y \):

\[ \text{Stack} \quad X \]

\[ \text{Y} \quad \text{X} \]
HW 1 will be posted tomorrow.

\texttt{SPAWN}

\texttt{spawn('program', file, arguments)} \leftarrow \texttt{win32}

\texttt{fork()}

\texttt{SPAWN}

\texttt{UNIX - fork()}

I/O Abstraction:

FILE I/O

You have a function to Read from a terminal.
Just one function to manage all terminal inputs:
Key: Read_Terminal (#)
1, 2, 3, 4

Similarly, Write_Terminal()

handle = open(Terminal 1)

read(handle)
write(handle)

Some integer value.

Read

3

$\{\text{data (lines in the OS private memory)}\}$

Read_Terminal.