Lecturer: Dr. Michael J. May
Kinneret College

Interrupts, Kernel Stack, Scheduler, SMT, `fork()`, Signals

10 November 2026
Lecture 4

Slides adapted from John Kubiatowicz (UC Berkeley)

Concept Review

Dual mode

Base and bound

- Absolute
- With adder

uPC (user PC)

Syscall

Interrupt

Trap/Exception

Address space translation

Topics for Today

- Interrupt Handling and Kernel Stack
- Processes and Scheduler
- Digging Deeper: SMT
- `fork()`
- Signals

Concepts today



How do we take interrupts safely?

Interrupt vector

- Limited number of entry points into kernel

Kernel interrupt stack

- Handler works regardless of state of user code

Interrupt masking

- Handler is non-blocking

Atomic transfer of control

- "Single instruction"-like to change the following:
  - Program counter
  - Stack pointer
  - Memory protection
  - Kernel/user mode

Transparent restartable execution

- User program does not know interrupt occurred

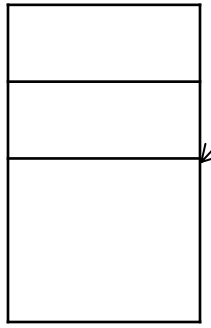
Interrupts on x86: Before

User-level Process

Code:

```
foo() {
   while (…) {
      x=x+1;
      y=y+1;
   }
}
```

Stack:



Registers

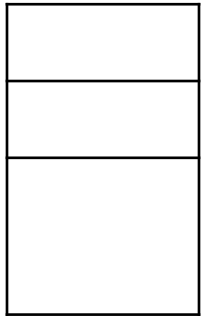
SS:ESP
CS:EIP
EFLAGS
Other registers: EAX, EBX

Kernel

Code:

```
Handler(){
   pusha
   …
}
```

Exception Stack:



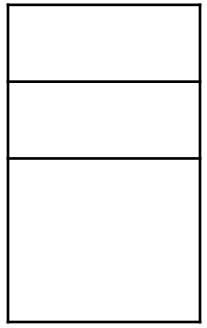
Interrupts on x86: During

User-level Process

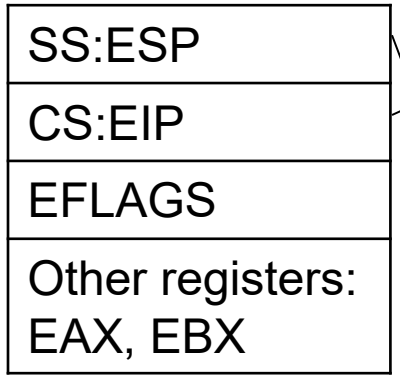
Code:

```
foo() {
   while (…) {
      x=x+1;
       y=y+1;
   }
}
```

Stack:



Registers

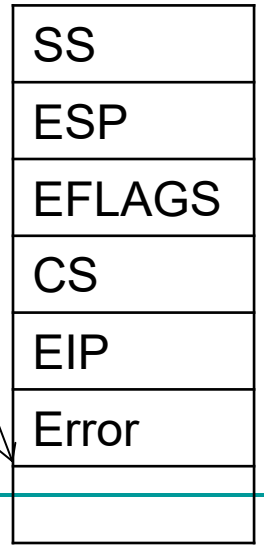


Kernel

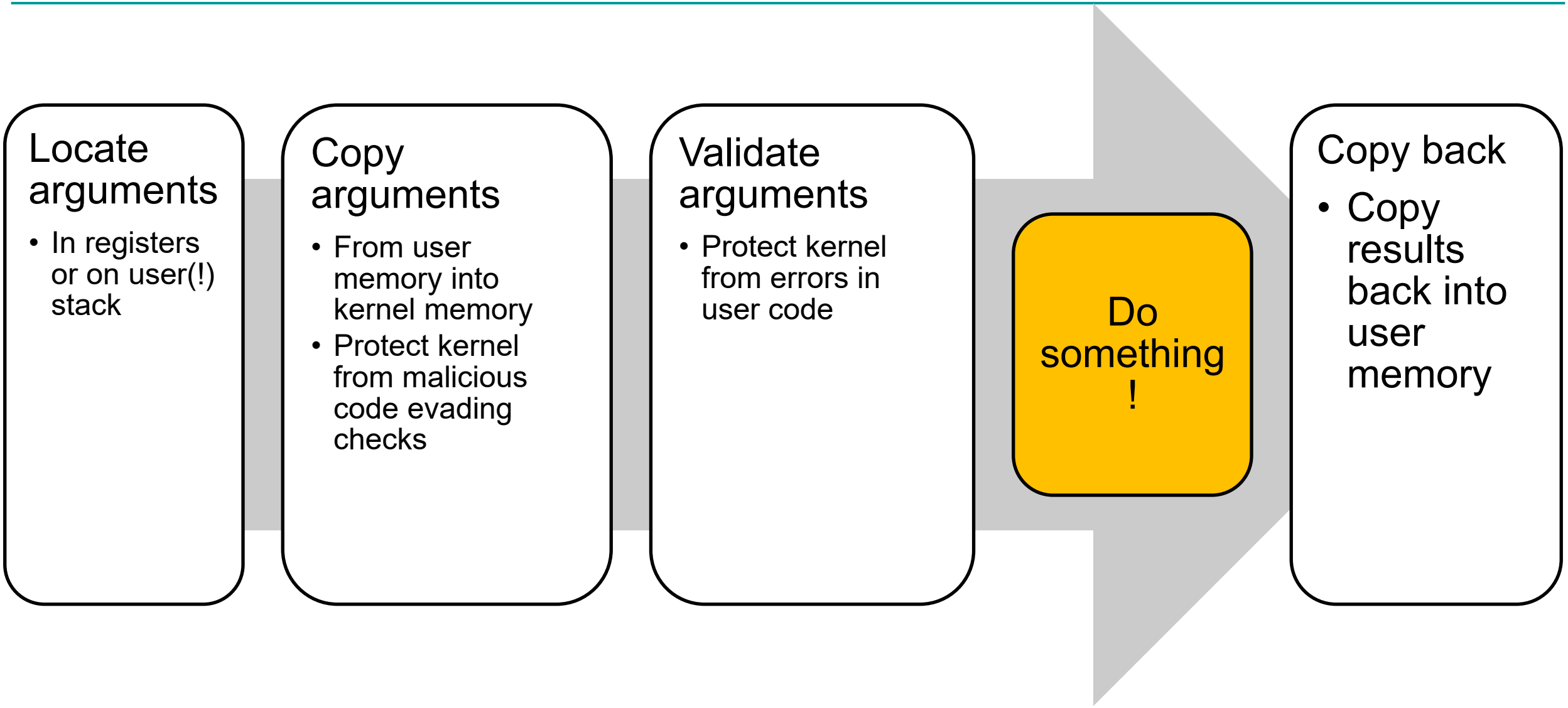
Code:

```
Handler(){
   pusha
   …
}
```

Exception Stack:



Kernel System Call Handler



So Far

- Interrupt Handling and Kernel Stack
- Processes and Scheduler
- Digging Deeper: SMT
- `fork()`
- Signals

Running Many Programs?

- We have the mechanism to
  - Switch between user processes and the kernel
  - Switch between user processes
  - Protect OS from user processes and processes from each other

Questions:

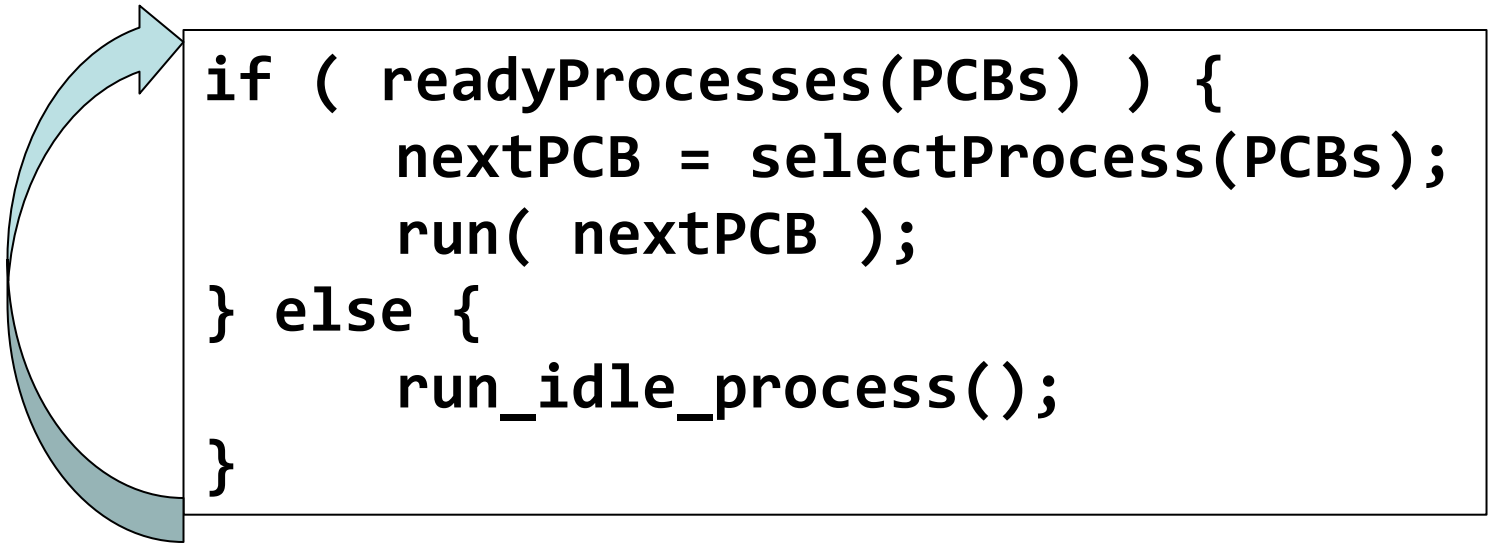
- How do we decide which user process to run?
- How does the OS record and manage user processes?
- How do we serialize the process and set it aside?
- How do we build a stack and heap for the kernel?
- How do we avoid wasting memory?

Process Control Block

- Kernel represents each process as a Process Control Block (PCB)
  - Status (running, ready, blocked, …)
  - Register state (when not running)
  - Process ID (PID), User, Executable, Priority, …
  - Execution time, …
  - Memory space, translation, …

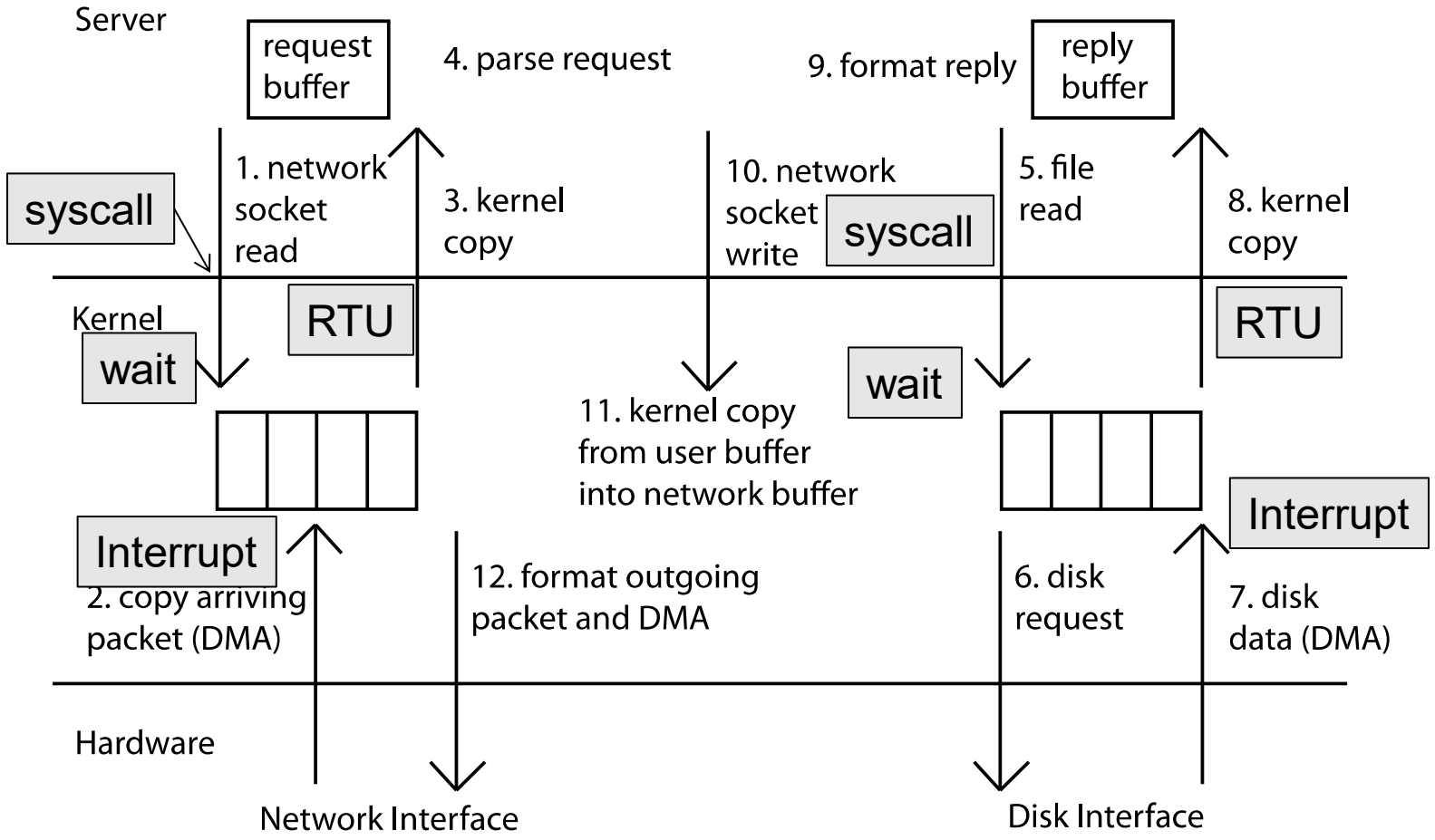
- Kernel Scheduler maintains a data structure containing the PCBs
- Scheduling algorithm selects the next one to run

Scheduler



```
if ( readyProcesses(PCBs) ) {
      nextPCB = selectProcess(PCBs);
      run( nextPCB );
} else {
      run_idle_process();
}
```

Putting it together: Web Server



So Far

- Interrupt Handling and Kernel Stack
- Processes and Scheduler
- Digging Deeper: SMT
- `fork()`
- Signals

Digging Deeper:

SMT

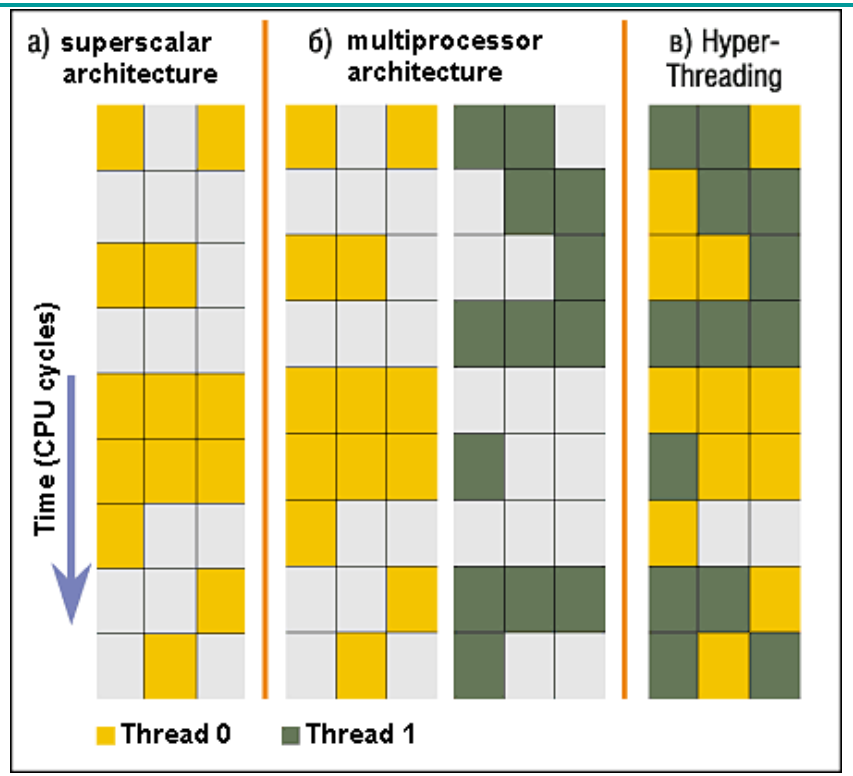
Simultaneous MultiThreading/Hyperthreading

Hardware technique

- Superscalar processors can execute multiple instructions that are independent.
- Hyperthreading duplicates register state to make a second "thread," allowing more instructions to run.

Can schedule each thread as if were separate CPU

- But, sub-linear speedup!



Colored blocks show instructions executed

Original technique called "Simultaneous Multithreading"

- http://www.cs.washington.edu/research/smt/index.html
- SPARC, Pentium 4/Xeon ("Hyperthreading"), Power 5

```
Item    Value
OS Name Microsoft Windows 10 Pro
Version 10.0.17134 Build 17134
Other OS Description    Not Available
OS Manufacturer Microsoft Corporation
System Name
System Manufacturer Dell Inc.
System Model    Latitude 5590
System Type x64-based PC
System SKU  0817
Processor   Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, 2112 Mhz, 4 Core(s), 8 Logical Processor(s)
BIOS Version/Date   Dell Inc. 1.3.2, 6/8/2018
```

So Far

- Interrupt Handling and Kernel Stack
- Processes and Scheduler
- Digging Deeper: SMT
- `fork()`
- Signals

Can a process create a process? How?



Option 1: Build a new one Windows

1. Allocate resources for new process – memory, virtual CPU, etc.
2. Assign task – an executable file
3. Assign ownership – user, parent process
4. Assign priority
5. Load task into an image
6. Mark it ready to run
7. Add it to ready queue

Who does steps 1-4?

Option 2: Copy existing one



1. Copy all attributes of existing process
2. Change parts necessary in new process – task, memory, virtual CPU, priority, ownership
3. Make modified process child of original process
4. Run modified process

Who does steps 1-3?

https://openclipart.org/detail/284231/chicken-with-eggs

Cloning



© 2024 Dav Pilkey

The UNIX/Linux way

Process ID

- Unique identity of process is the “process ID” (or pid).
- `fork()` creates a *copy* of current process with a new pid

State of original process duplicated in both Parent and Child!

- Memory, File Descriptors (next topic), etc

Return value from `fork()`: integer

- When > 0
  - Running in (original) Parent process
  - Return value is pid of new child
- When = 0
  - Running in new Child process
- When < 0
  - Error! Must handle somehow, running in original process

fork1.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

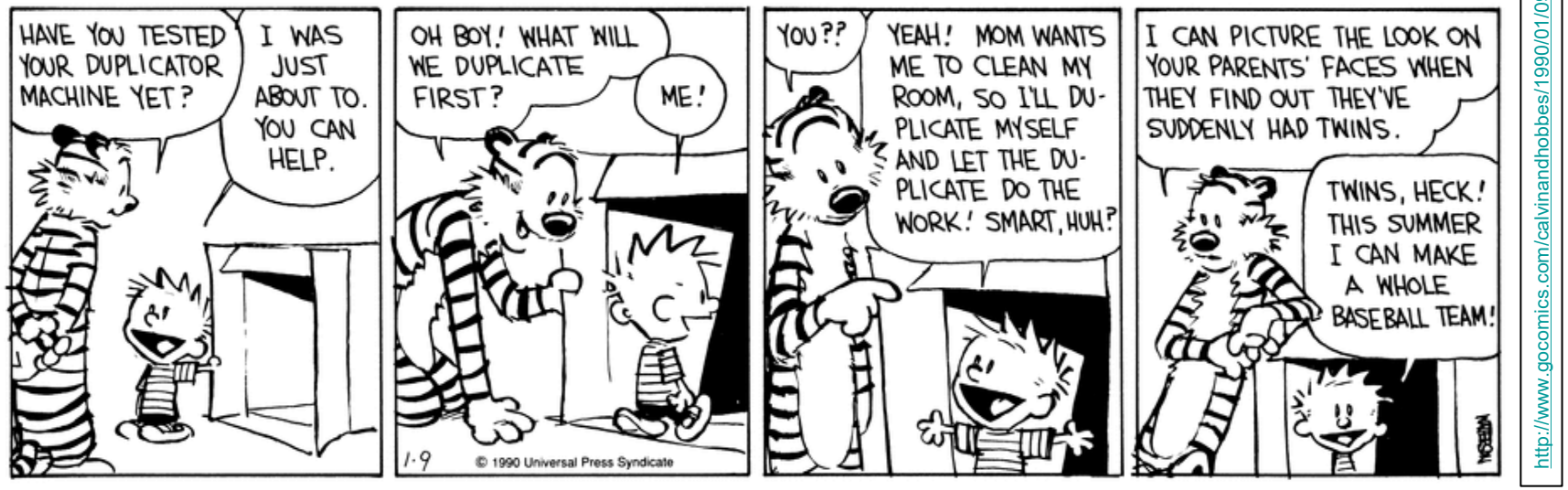
#define BUFSIZE 1024
int main(int argc, char *argv[])
{
  char buf[BUFSIZE];
  size_t readlen, writelen, slen;
  pid_t cpid, mypid;
  pid_t pid = getpid();         /* get current processes PID */
  printf("Parent pid: %d\n", pid);
  cpid = fork();
  if (cpid > 0) {                    /* Parent Process */
    mypid = getpid();
    printf("[%d] parent of [%d]\n", mypid, cpid);
  }  else if (cpid == 0) {   /* Child Process */
    mypid = getpid();
    printf("[%d] child\n", mypid);
  } else {
    perror("Fork failed");
    exit(1);
  }
  exit(0);
}
```

How to tell them apart?



- http://www.gocomics.com/calvinandhobbes/1990/01/09
- http://www.gocomics.com/calvinandhobbes/1990/01/11

Fork imagined (© Bill Waterson)



http://www.gocomics.com/calvinandhobbes/1990/01/09

http://www.gocomics.com/calvinandhobbes/1990/01/11

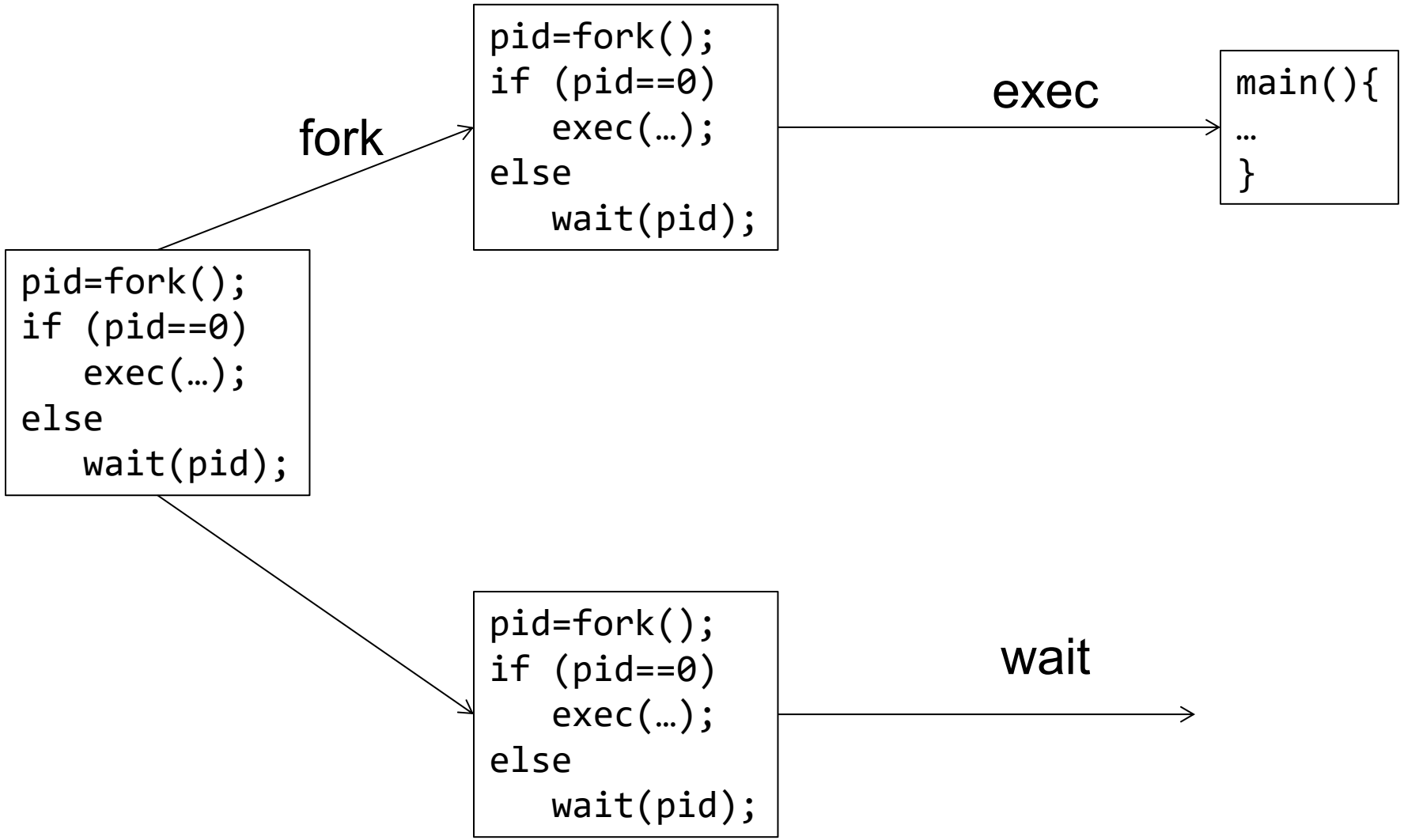
https://meaww.com/orphan-black-the-next-chapter-tatiana-maslany-teases-new-clones-fate-of-old-characters-10-years



fork2.c

```
int status;
…
cpid = fork();
if (cpid > 0) {                    /* Parent Process */
       mypid = getpid();
       printf("[%d] parent of [%d]\n", mypid, cpid);
       tcpid = wait(&status);
       printf("[%d] bye %d(%d)\n", mypid, tcpid, status);
}  else if (cpid == 0) {           /* Child Process */
       mypid = getpid();
       printf("[%d] child\n", mypid);
}
…
```

Fork and Wait



Shell

- A shell is a job control system
  - Allows programmer to create and manage a set of programs to do some task
  - Windows, MacOS, Linux all have shells

- Example: to compile a C program

```
cc –c sourcefile1.c
cc –c sourcefile2.c
ln –o program sourcefile1.o sourcefile2.o
./program
```

process-race.c

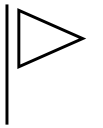
```
int i;
cpid = fork();
if (cpid > 0 ) {
   mypid = getpid();
   printf("[%d] parent of [%d]\n", mypid, cpid);
   for (i = 0; i< 100; i++) {
      printf("[%d] parent: %d\n", mypid, i);
      // sleep(1);
   }
}
else if (cpid == 0) {
   mypid = getpid();
   printf("[%d] child\n", mypid);
   for (i = 0; i > -100; i--)  {
      printf("[%d] child: %d\n", mypid, i);
      // sleep(1);
   }
}
```

What does this program print?
Does it change if you add the sleep?

So Far

- Interrupt Handling and Kernel Stack
- Processes and Scheduler
- Digging Deeper: SMT
- `fork()`
- Signals

Signals: Simple Messaging



- Signal: Like a software interrupt
  - Simple (integer) message you can send a process
  - Interrupts normal execution
- Sent by:
  - One process to another (syscall or via `kill` program)
  - OS to a process (due to event)

- Sample signals:

SIGINT	Interrupt from keyboard (CTRL+C)
SIGQUIT	Quit from keyboard (CTRL+\)
SIGCONT	Continue a program
SIGKILL	Kill signal
SIGTTIN	Background process tries to read from STDIN

Signals – infloop.c

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

void signal_callback_handler(int signum)
{
  printf("Caught signal %d - phew!\n",signum);
  exit(1);
}

int main() {
  signal(SIGINT, signal_callback_handler);
  while (1) {}
}
```

How do you stop this?

Look at `top`

Some more signals – signals-sample.c

```
void signal_handler(int sig) {
  switch (sig) {
    case SIGINT:
      printf("Received SIGINT (Interrupt from keyboard), signal number: %d\n", sig);
      break;
    case SIGTERM:
      printf("Received SIGTERM (Termination signal), signal number: %d\n", sig);
      break;
    case SIGQUIT:
      printf("Received SIGQUIT (Quit from keyboard), signal number: %d\n", sig);
      break;
    default:
      printf("Received signal number: %d\n", sig);
  }
}
```



signals-sample.c

```
int main() {
    if (signal(SIGINT, signal_handler) == SIG_ERR) {
        perror("Error registering SIGINT handler");
        exit(1);
    }
    if (signal(SIGTERM, signal_handler) == SIG_ERR) {
        perror("Error registering SIGTERM handler");
        exit(1);
    }
    if (signal(SIGQUIT, signal_handler) == SIG_ERR) {
        perror("Error registering SIGQUIT handler");
        exit(1);
    }
    printf("Running... Press Ctrl+C to send SIGINT, or send SIGTERM or SIGQUIT to this process.\n");
    while (1) {
        sleep(1);
    }
    return 0;
}
```

An aside about System Calls (syscalls)

What is a syscall?

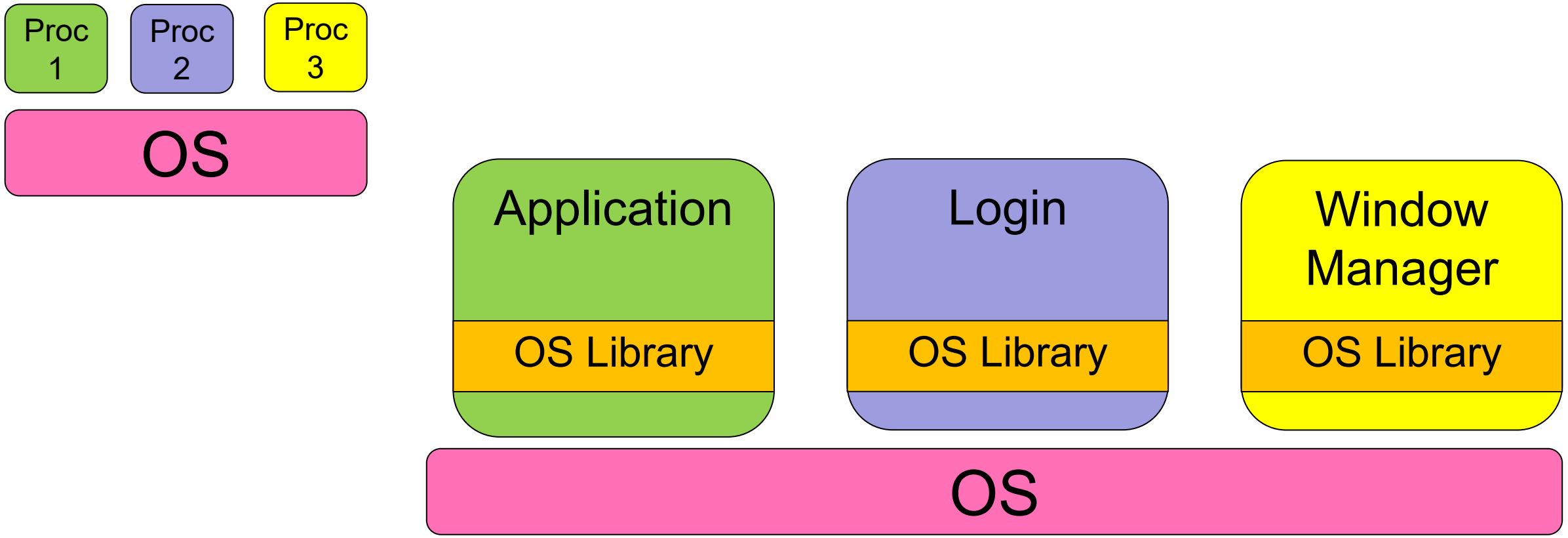
- Applications request services from the operating system via a ***syscall***, but …
- I've been writing applications and never saw a "syscall" in code?

Why?

Syscalls are buried in the programming language runtime library (ex. `libc.a`)

Layering at work

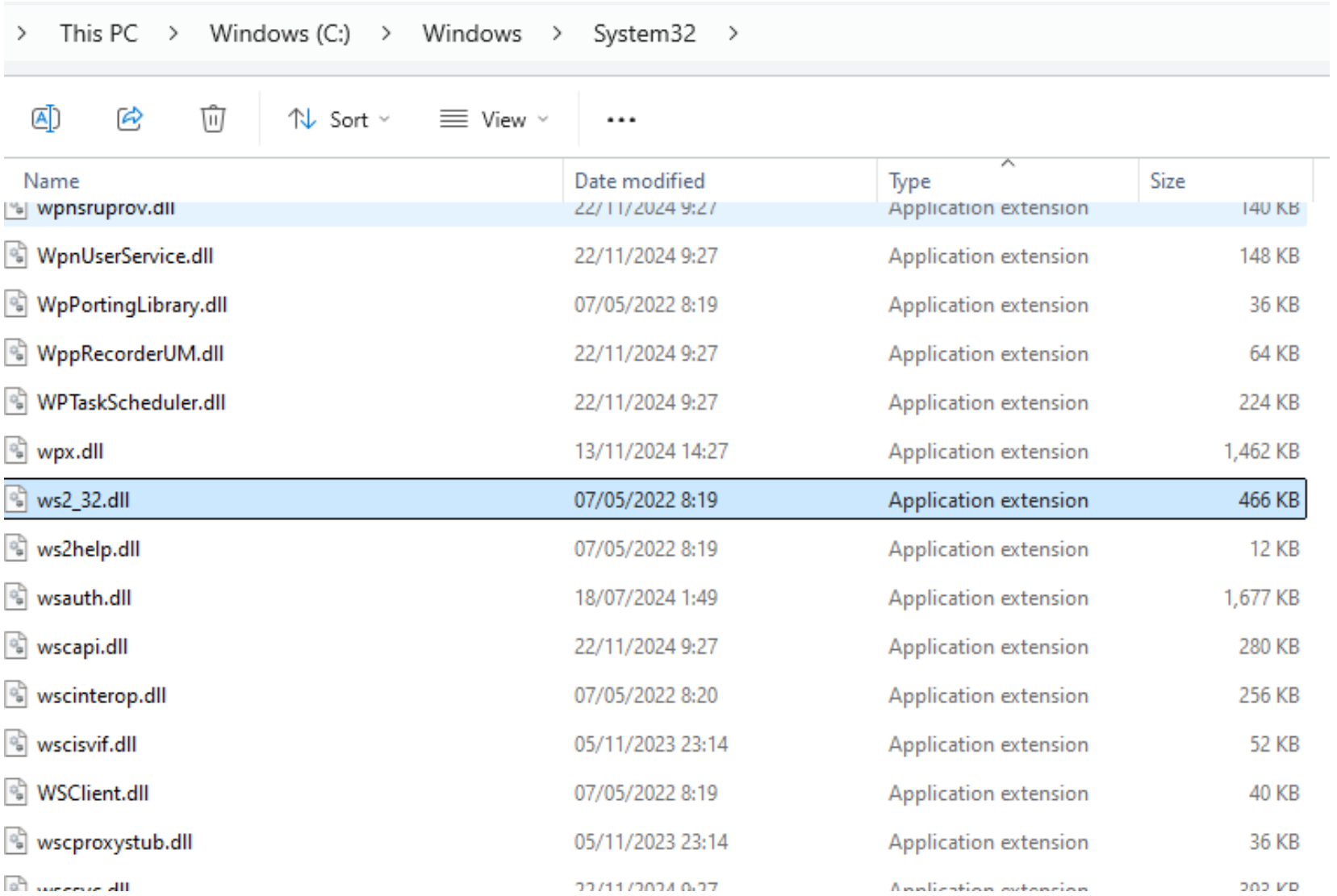
OS run-time library



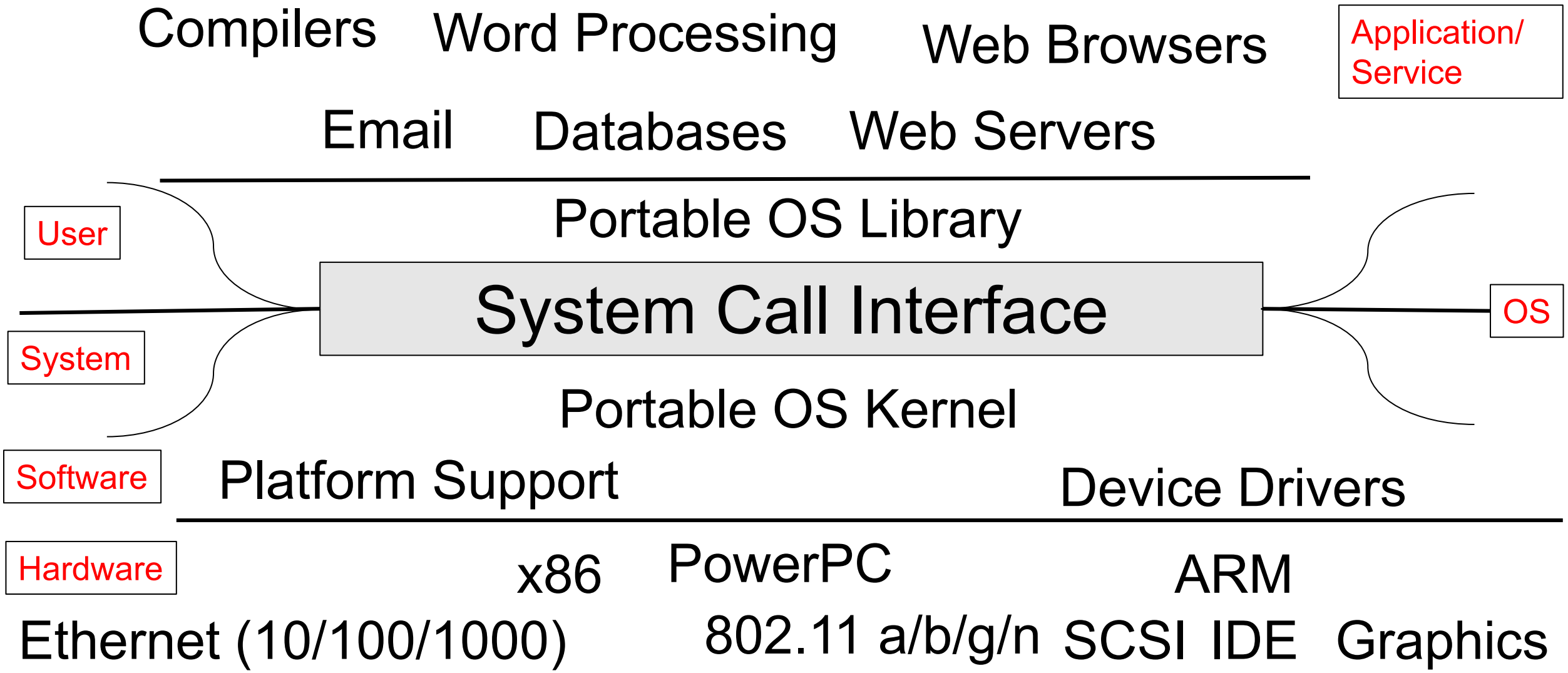
In WSL (Ubuntu)

```
mjmay@RAMOT:/lib/x86_64-linux-gnu$ ls
Mcrt1.o                                 libcurl-gnutls.so.4.7.0                 libjack.so.0.1.0                        libpython3.10.so.1.0
Scrt1.o                                 libcurl.so.4                            libjacknet.so.0                         libquadmath.so.0
audit                                   libcurl.so.4.7.0                        libjacknet.so.0.1.0                     libquadmath.so.0.0.0
avahi                                   libcurses.a                             libjackserver.so.0                      libraw1394.so.11
awk                                     libcurses.so                            libjackserver.so.0.1.0                  libraw1394.so.11.1.0
bfd-plugins                             libdatrie.so.1                          libjansson.so.4                         libreadline.so.8
caca                                    libdatrie.so.1.4.0                      libjansson.so.4.13.0                    libreadline.so.8.1
crt1.o                                  libdb-5.3.so                            libjavascriptcoregtk-4.0.so.18          libresolv.a
crti.o                                  libdbus-1.so.3                          libjavascriptcoregtk-4.0.so.18.25.12    libresolv.so
crtn.o                                  libdbus-1.so.3.19.13                    libjbig.so.0                            libresolv.so.2
dri                                     libdconf.so.1                           libjpeg.so.8                            librom1394.so.0
e2fsprogs                               libdconf.so.1.0.0                       libjpeg.so.8.2.2                        librom1394.so.0.3.0
enchant-2                               libdebconfclient.so.0                   libjson-c.so.5                          librsvg-2.so.2
engines-3                               libdebconfclient.so.0.0.0               libjson-c.so.5.1.0                      librsvg-2.so.2.48.0
gawk                                    libdeflate.so.0                         libjson-glib-1.0.so.0                   librt.a
gconv                                   libdevmapper.so.1.02.1                  libjson-glib-1.0.so.0.600.6             librt.so.1
gcrt1.o                                 libdl.a                                 libk5crypto.so.3                        librtmp.so.1
gdk-pixbuf-2.0                          libdl.so.2                              libk5crypto.so.3.1                      libsamplerate.so.0
gedit                                   libdns-9.18.28-0ubuntu0.22.04.1-Ubuntu.so  libkeyutils.so.1                        libsamplerate.so.0.2.2
gio                                     libdns-export.so.1110                   libkeyutils.so.1.9                      libsasl2.so.2
girepository-1.0                        libdns-export.so.1110.0.2               libkmod.so.2                            libsasl2.so.2.0.25
glib-2.0                                libdrm.so.2                             libkmod.so.2.3.7                        libseccomp.so.2
graphviz                                libdrm.so.2.4.0                         libkrb5.so.3                            libseccomp.so.2.5.3
```

In Windows 11



A Narrow Waist





© Reuters

https://www.independent.co.uk/news/world/asia/hundreds-of-cars-caught-up-in-beijing-traffic-during-golden-week-a6685961.html

Conclusion

- Interrupt Handling and Kernel Stack
- Processes and Scheduler
- Digging Deeper: SMT
- `fork()`
- Signals