

Signals

(Operating System)

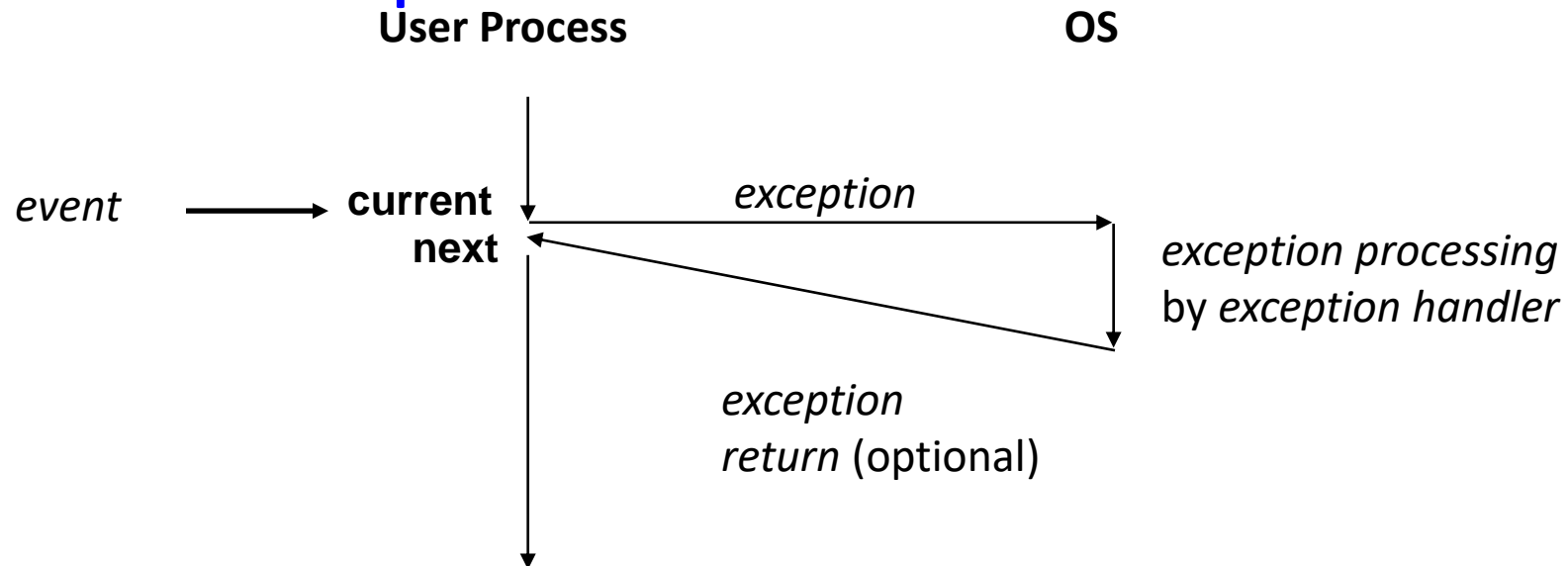


Deepika H V

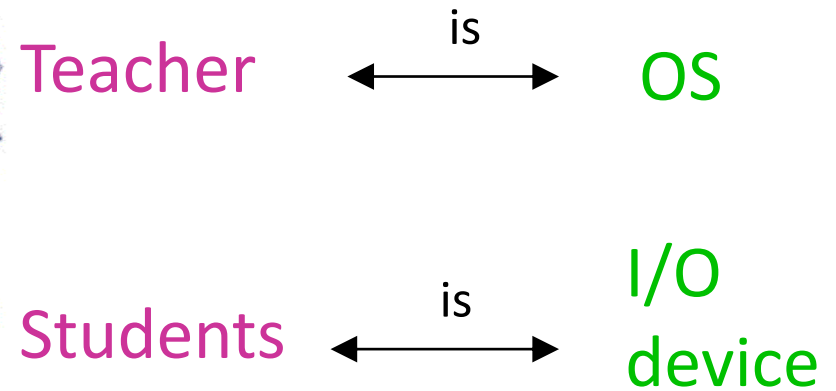
C-DAC Bengaluru
(deepikahv@cdac.in)

Interrupt

- An unexpected event that causes a processor to (temporarily) **transfer control** to another program, or function.
- When that function **completes**, control is (typically) **returned** to the interrupted process, which resumes from the point it was interrupted



Interrupt



- **Hardware Interrupts**
 - asynchronous entities
 - typically employed to provide an effective means for a system to react to outside stimuli.
- **Software Interrupts (Exceptions)**
 - Has both synch and asynch
 - Exceptions generated by processes.
 - Caused by events that occur as result of executing an instruction

Hardware Interrupts

- **I/O interrupts**
 - hitting ctl-c at the keyboard
 - arrival of a packet from a network
 - arrival of a data sector from a disk
- **Hard reset interrupt**
 - hitting the reset button
- **Soft-reset interrupts**
 - hitting ctl-alt-delete on a PC

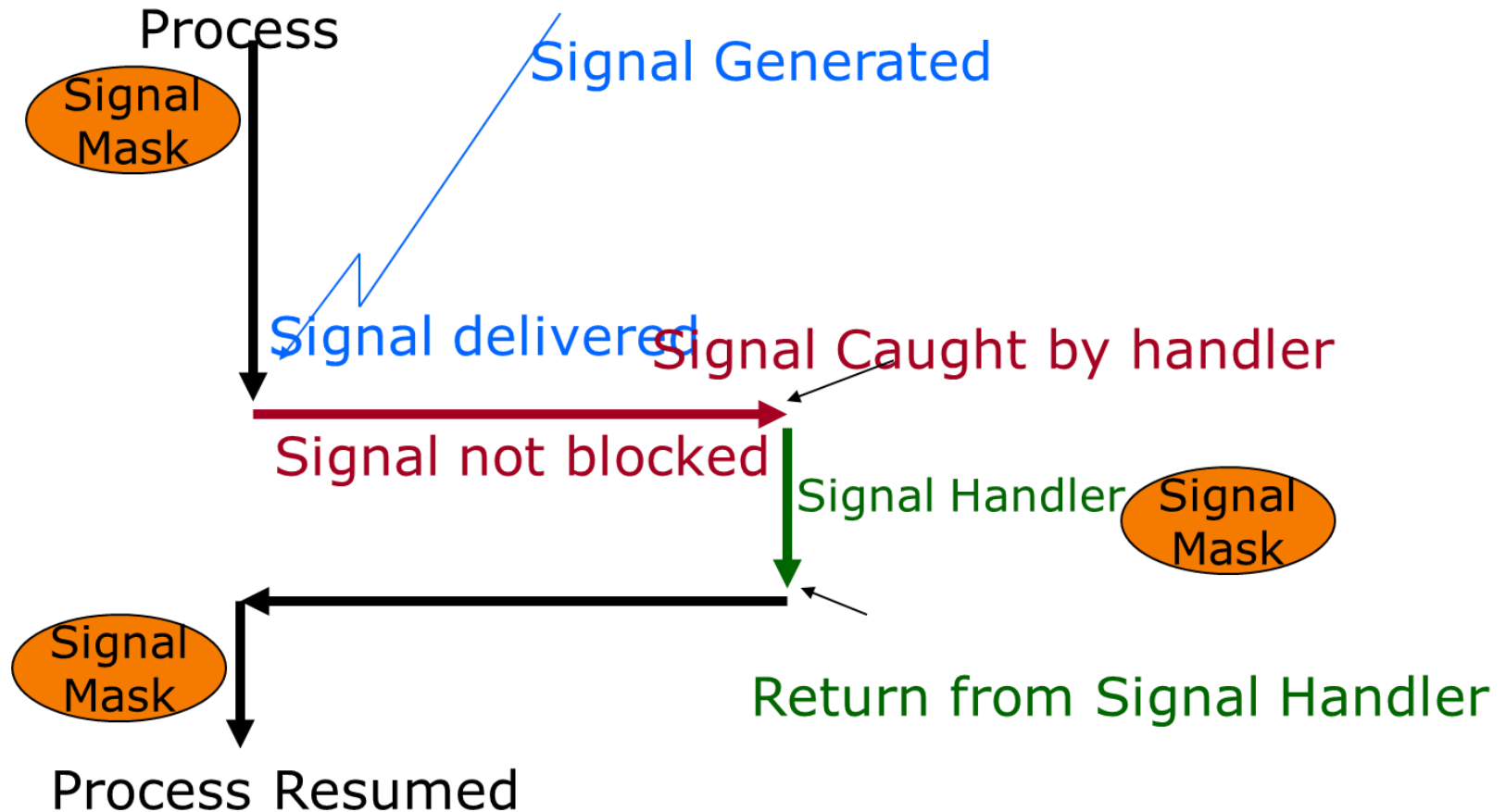
Software Interrupts

- **Traps**
 - Intentional - system calls, breakpoint traps, special instruction
 - Returns control to “next” instruction
- **Faults**
 - Unintentional but possibly recoverable - **page faults**.
 - Either **re-executes** faulting instruction.
- **Aborts**
 - unintentional and **unrecoverable** -parity error, machine check.
 - **Aborts** current program

Signals

- **A notification of an event**
 - Event gains attention of the OS
 - OS stops the current process, sending it a signal
 - Signal handler executes to completion
 - Application process resumes where it left off
- **Different signals are identified by small integer ID's**

ID	Name	Default Action	Corresponding Event
2	SIGINT	Terminate	Interrupt from keyboard (ctrl-c)
9	SIGKILL	Terminate	Kill program (cannot override or ignore)
11	SIGSEGV	Terminate & Dump	Segmentation violation
14	SIGALRM	Terminate	Timer signal
17	SIGCHLD	Ignore	Child stopped or terminated



Signal Mask

- Process can temporarily prevent signal from being delivered by **blocking** it.
- **Signal Mask** contains a set of signals currently blocked.
- **Important! Blocking a signal is different from ignoring signal. Why?**
 - When a process blocks a signal, the OS does not deliver signal until the process unblocks the signal
 - A **blocked** signal is not delivered to a process until it is unblocked.
 - When a process ignores signal, signal is delivered and the process handles it by throwing it away.

Sending a Signal

- **Kernel sends (delivers) a signal to a destination process by updating some state in the context of the destination process**
- **Kernel sends a signal for one of the following reasons:**
 - Kernel has detected a system event such as divide by zero (SIGFPE) or termination of a child process (SIGCHLD)
 - Another process has invoked the kill system call to explicitly request that the kernel send a signal to the destination process

Receiving

- A destination process receives a signal when it is forced by the kernel to react in some way to the delivery of the signal
- Five possible ways to react:
 - Ignore the signal (do nothing)
 - Terminate the process
 - Temporarily stop the process from running
 - Continue a stopped process (let it run again)
 - Catch the signal by executing a user-level function called a signal handler

Predefined Signals

```
$ kill -l
```

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL
5) SIGTRAP	6) SIGABRT	7) SIGBUS	8) SIGFPE
9) SIGKILL	10) SIGUSR1	11) SIGSEGV	12) SIGUSR2
13) SIGPIPE	14) SIGALRM	15) SIGTERM	17) SIGCHLD
18) SIGCONT	19) SIGSTOP	20) SIGTSTP	21) SIGTTIN
22) SIGTTOU	23) SIGURG	24) SIGXCPU	25) SIGXFSZ
26) SIGVTALRM	27) SIGPROF	28) SIGWINCH	29) SIGIO
30) SIGPWR	31) SIGSYS	34) SIGRTMIN	35) SIGRTMIN+1
36) SIGRTMIN+2	37) SIGRTMIN+3	38) SIGRTMIN+4	39) SIGRTMIN+5
40) SIGRTMIN+6	41) SIGRTMIN+7	42) SIGRTMIN+8	43) SIGRTMIN+9
44) SIGRTMIN+10	45) SIGRTMIN+11	46) SIGRTMIN+12	47) SIGRTMIN+13
48) SIGRTMIN+14	49) SIGRTMIN+15	50) SIGRTMAX-14	51) SIGRTMAX-13
52) SIGRTMAX-12	53) SIGRTMAX-11	54) SIGRTMAX-10	55) SIGRTMAX-9
56) SIGRTMAX-8	57) SIGRTMAX-7	58) SIGRTMAX-6	59) SIGRTMAX-5
60) SIGRTMAX-4	61) SIGRTMAX-3	62) SIGRTMAX-2	63) SIGRTMAX-1
64) SIGRTMAX			

Signal via Keyboard

- **Ctrl-c -> 2/SIGINT signal**
 - Default handler exits process
- **Ctrl-z -> 20/SIGTSTP signal**
 - Default handler suspends process
- **Ctrl-\ -> 3/SIGQUIT signal**
 - Default handler exits process
- **Check using stty -a**

Signal via Commands

- **kill -signal pid**
 - Send a signal of type signal to the process with id pid
 - Can specify either signal type name (-SIGINT) or number (-2)
- **No signal type name or number specified => sends 15/SIGTERM signal**
 - Default 15/SIGTERM handler exits process
- **Examples**
 - kill -2 1234
 - kill -SIGINT 1234

Signal via function call – raise()

- **int raise(int iSig);**
 - Commands OS to send a signal of type iSig to current process
 - Returns 0 to indicate success, non-0 to indicate failure
- **Example**

```
int ret = raise(SIGINT); /* Process commits suicide. */
assert(ret != 0);      /* Shouldn't get here. */
```

```
#include<stdio.h>
#include<signal.h>
```

```
void main()
{
    printf(" the process id is %d\n",getpid());
    raise(SIGINT);
    printf(" the parent process id is %d\n",getppid());

}
```


Signal via function call – kill()

- **int kill(pid_t iPid, int iSig);**
 - Sends a iSig signal to the process whose id is iPid
 - Equivalent to raise(iSig) when iPid is the id of current process
- **Example**

```
pid_t iPid = getpid(); /* Process gets its id.*/  
kill(iPid, SIGINT);
```

```
#include<stdio.h>
#include<signal.h>

void main()
{
    int ret;
    ret = fork();
    if(ret == 0)
    {
        printf(" the process id is %d\n",getpid());
        printf(" the parent process id is %d\n",getppid());
        for(;;)
            printf("Looping in child process\n");
    }
    else
    {
        kill(ret,SIGINT);
    }
}
```

Signal via function call - signal

- **sighandler_t signal(int iSig, sighandler_t pfHandler);**
 - Installs function pfHandler as the handler for signals of type iSig
 - pfHandler is a function pointer:
- **Returns the old handler on success, SIG_ERR on error**
- **pfHandler is invoked whenever process receives a signal of type iSig**

```
int main(void) {  
    void (*pfRet) (int);  
    pfRet = signal(SIGINT, SIG_IGN);  
    ...  
}
```

```
...
static FILE *psFile; /* Must be global. */
static void cleanup(int iSig) {
    fclose(psFile);
    remove("tmp.txt");
    exit(EXIT_FAILURE);
}
int main(void) {
    void (*pfRet)(int);
    psFile = fopen("temp.txt", "w");
    pfRet = signal(SIGINT, cleanup);
    ...
    raise(SIGINT);
    return 0; /* Never get here. */
}
```

Terminologies

- Signal generated
- Signal Delivered
- Lifetime
- Pending
- Signal caught
- Signal ignored
- Signal blocked

THANK YOU