Recitation 8: Signals

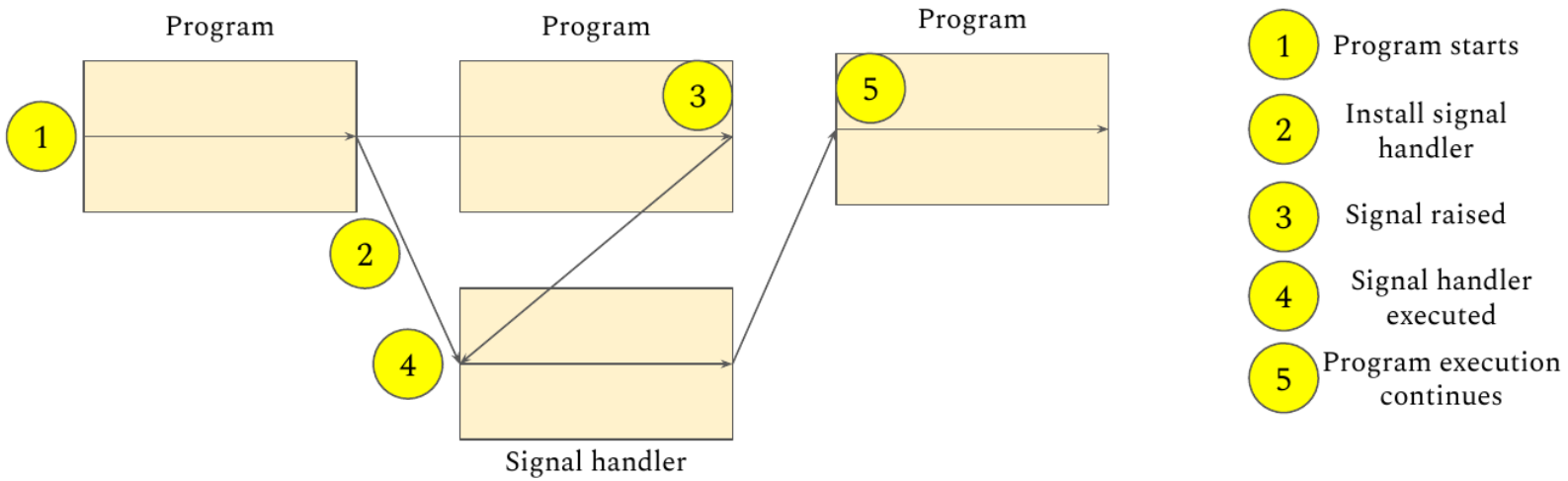
CSCI 4061: Introduction to Operating Systems

Overview

- Signals
- Exercises

Signals

- Software notification to a process of an event
- A signal that is generated but not yet delivered is in **pending** state (block/unblock - later slides)
- A process **catches** a signal and executes corresponding **signal handler**



Common signals

Signal	**Description**	**Default action**
SIGINT	Interactive attention signal (Ctrl + C)	Abnormal termination
SIGTERM	Termination	Abnormal termination
SIGSEGV	Invalid memory reference	Implementation dependent
SIGKILL	Terminated (cannot be caught or ignored)	Abnormal termination
SIGTSTP	Terminal Stop (Ctrl + Z)	Stop
SIGUSR1	User defined signal 1	Abnormal termination
SIGUSR2	User-defined signal 2	Abnormal termination

Signal Handling

```
1 #include <signal.h>
2
3 int sigaction(int signum, const struct sigaction *act,
4                struct sigaction *oldact);
```

Set signal handling info corresponding to signum signal

- `signum`: macro of the signal to be handled (SIGINT, SIGTERM, ...)
- `act`: structure with info on how to handle the signal
- `oldact`: usually NULL, if not, then previous signal handling info stored

Signal Handling(cont.)

```
1 struct sigaction {
2     void (*sa_handler)(int); // SIG_DFL, SIG_IGN,
    signal handling function
3     void (*sa_sigaction)(int, siginfo_t *, void *); //
     another way to handle signal
4     sigset_t sa_mask; // mask of signals to be blocked
     during signal handler execution
5     int sa_flags; // modify behavior of signals,
    usually 0
6     void (*sa_restorer)(void); // not intended for app
     use
7 };
```

Underlined fields relevant

Returns 0 on success and -1 on failure

Signal Blocking

- Signals are not delivered to the process
- Stays in **pending** state unless unblocked
- **Signal mask**: represent signals currently blocked
- **Signal set**: uses `sigset_t` datatype to store information on signals to be blocked

Operations on sigset_t

```
#include <signal.h>

int sigemptyset(sigset_t *set); // clear all signals set
int sigfillset(sigset_t *set); // set all signals
int sigaddset(sigset_t *set, int signum); // set signum in set
int sigdelset(sigset_t *set, int signum); // unset signum in set
```

All above: Returns 0 on success and -1 on failure

```
int sigismember(const sigset_t *set, int signum); // check if signum is set
```

Returns 1 on finding signum, 0 on not finding and -1 on failure

Signal Blocking/Unblocking

- Fetch and/or change the signal mask of the calling thread

```
1 #include <signal.h>
2
3 int sigprocmask(int how, const sigset_t *set, sigset_t
      *oldset);
```

Parameters:

- how:
  - SIG_BLOCK: The set of signals in set is blocking
  - SIG_UNBLOCK: The set of blocking signals in set are removed and unblocked
  - SIG_SETMASK: Set current signal mask to given set
- set: Change the current signal mask to given set
- oldset: Usually NULL, if not, the previous signal mask is stored in oldset

Returns 0 on success and -1 on failure

- Once a blocked signal is unblocked, its action takes effect immediately

Exercise

In `samples/pgm1.c`, a signal handler ensures graceful exit on using Ctrl+C. In `samples/pgm2.c`, the Ctrl+Z is blocked and unblocked using masks. For this exercise, you will use the code in `pgm1.c` and `pgm2.c` to handle Ctrl+C.

1. First, you will install a signal handler for Ctrl + C where the loop variable is set.
2. Second, you will disable the SIGINT signal when counter is 0 and enable it back when the counter variable in the code reaches 5.

Note the sleep in the code.

Deliverables

Submit the **.zip** file of the exercises folder and submit to Gradescope by Aug 1st, 2025, 11:59 PM.