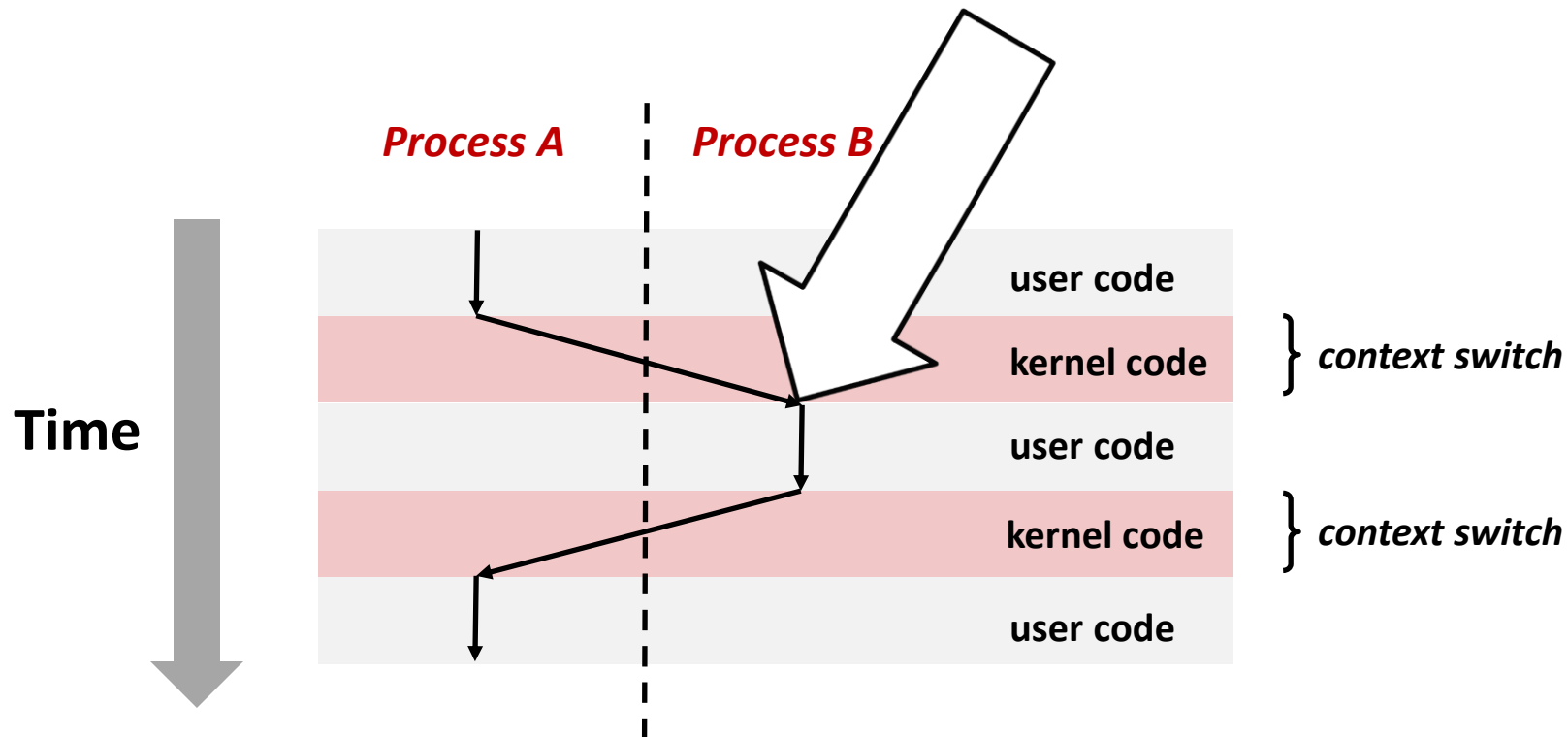


Exceptional Control Flow:

Signals: Receiving

Receiving Signals

- Suppose kernel is returning from an exception handler and is ready to pass control to process p



Receiving Signals

- Suppose kernel is returning from an exception handler and is ready to pass control to process p
- Kernel computes $\text{pnb} = \text{pending} \ \& \ \sim\text{blocked}$
 - The set of pending nonblocked signals for process p

Receiving Signals

- Suppose kernel is returning from an exception handler and is ready to pass control to process p
- Kernel computes $\text{pnb} = \text{pending} \ \& \ \sim\text{blocked}$
 - The set of pending nonblocked signals for process p
- If $(\text{pnb} == 0)$
 - Pass control to next instruction in the logical flow for p

Receiving Signals

- Suppose kernel is returning from an exception handler and is ready to pass control to process p
- Kernel computes $\mathbf{pnb} = \mathbf{pending} \ \& \ \sim\mathbf{blocked}$
 - The set of pending nonblocked signals for process p
- If $(\mathbf{pnb} == 0)$
 - Pass control to next instruction in the logical flow for p
- Else
 - Choose least nonzero bit k in \mathbf{pnb} and force process p to *receive* signal k
 - The receipt of the signal triggers some *action* by p
 - Repeat for all nonzero k in \mathbf{pnb}
 - Pass control to next instruction in logical flow for p

Default Actions

- Each signal type has a predefined *default action*, which is one of:
 - The process terminates
 - The process stops until restarted by a SIGCONT signal
 - The process ignores the signal

Installing Signal Handlers

- The `signal` function modifies the default action associated with the receipt of signal `signum`:
 - `handler_t *signal(int signum, handler_t *handler)`

Installing Signal Handlers

- The `signal` function modifies the default action associated with the receipt of signal `signum`:
 - `handler_t *signal(int signum, handler_t *handler)`
- Different values for `handler`:
 - `SIG_IGN`: ignore signals of type `signum`
 - `SIG_DFL`: revert to the default action on receipt of signals of type `signum`
 - Otherwise, `handler` is the address of a user-level *signal handler*
 - Called when process receives signal of type `signum`
 - Referred to as *“installing”* the handler
 - Executing handler is called *“catching”* or *“handling”* the signal
 - When the handler executes its return statement, control passes back to instruction in the control flow of the process that was interrupted by receipt of the signal

Signal Handling Example

```
void sigint_handler(int sig) /* SIGINT handler */
{
    printf("So you think you can stop the bomb with ctrl-c, do you?\n");
    sleep(2);
    printf("Well...");
    fflush(stdout);
    sleep(1);
    printf("OK. :-)\n");
    exit(0);
}

int main()
{
    /* Install the SIGINT handler */
    if (signal(SIGINT, sigint_handler) == SIG_ERR)
        unix_error("signal error");

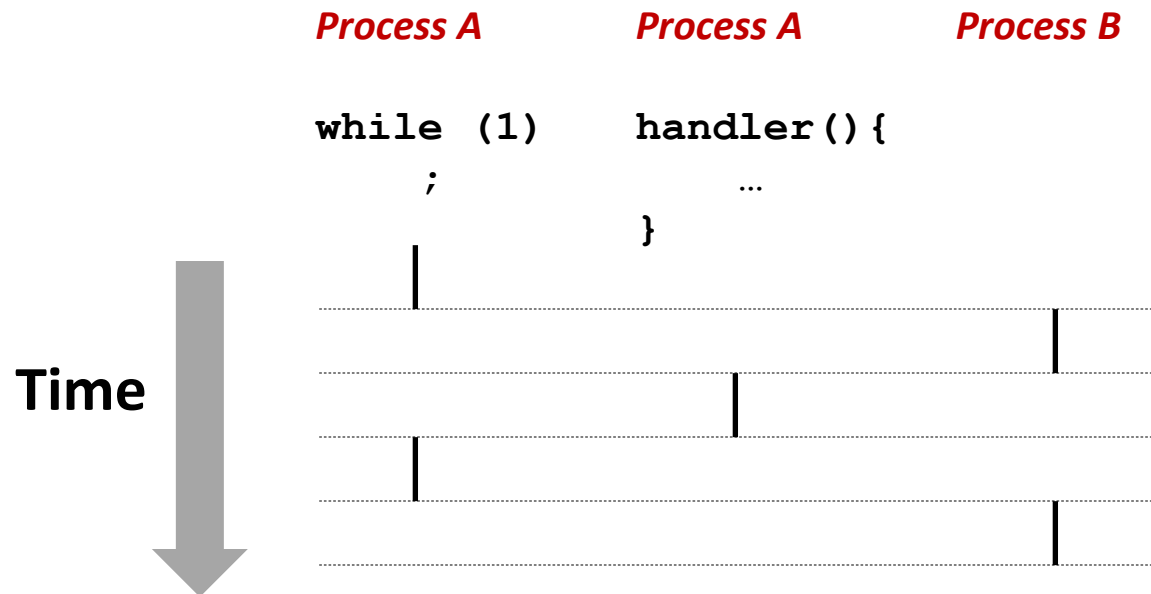
    /* Wait for the receipt of a signal */
    pause();

    return 0;
}
```

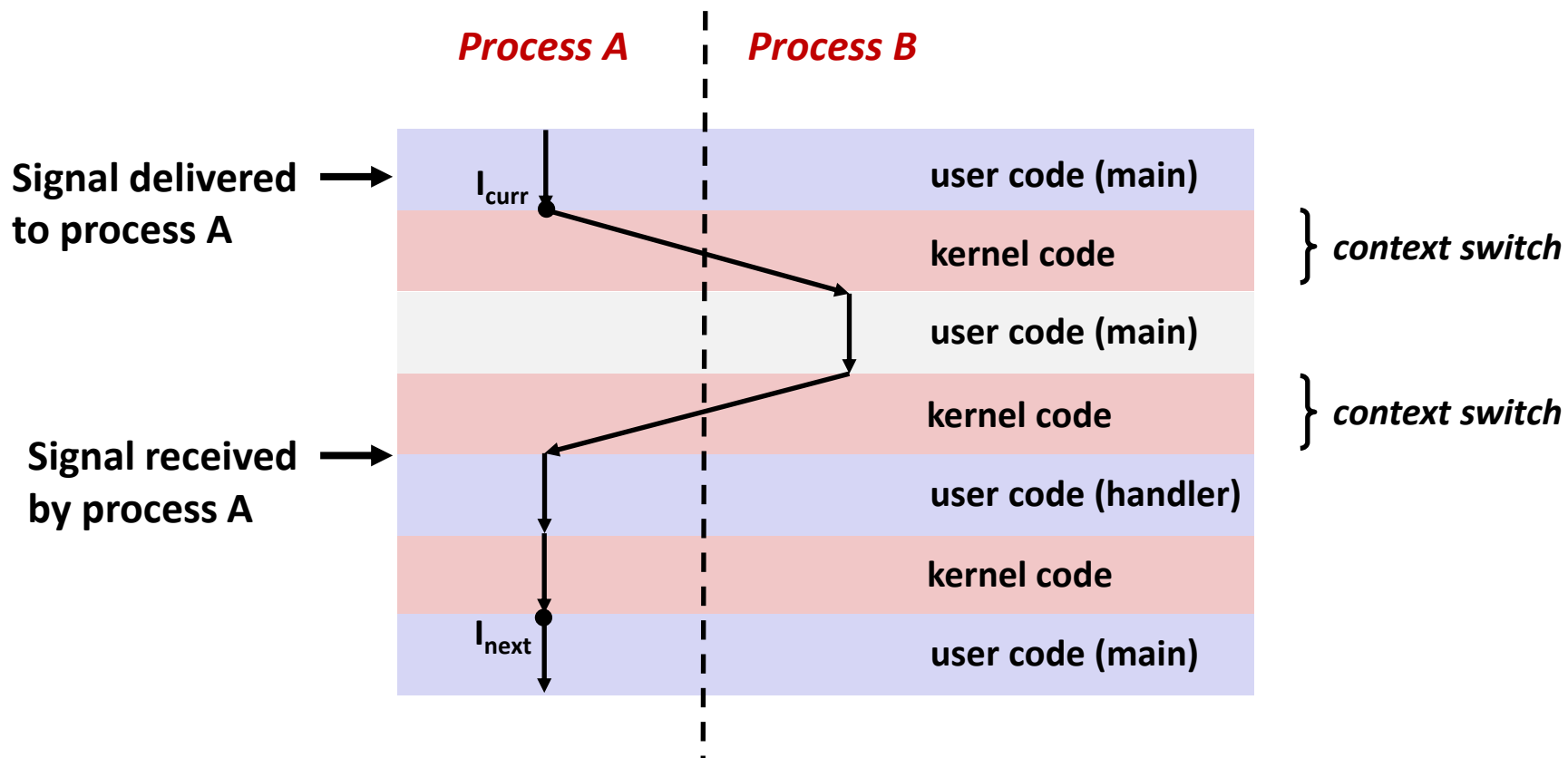
sigint.c

Signals Handlers as Concurrent Flows

- A signal handler is a separate logical flow (not process) that runs concurrently with the main program

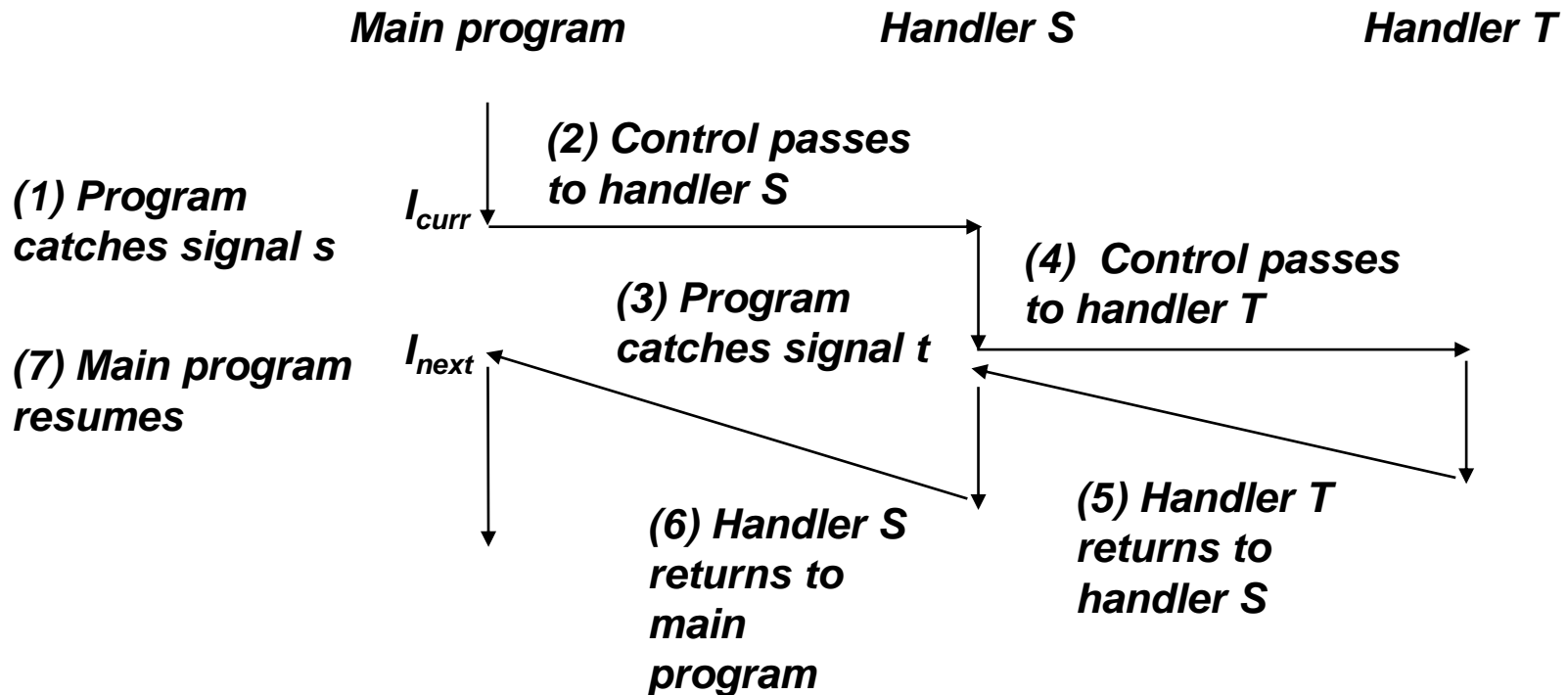


Another View of Signal Handlers as Concurrent Flows



Nested Signal Handlers

- Handlers can be interrupted by other handlers



Blocking and Unblocking Signals

■ Implicit blocking mechanism

- Kernel blocks any pending signals of type currently being handled.
- E.g., A SIGINT handler can't be interrupted by another SIGINT

■ Explicit blocking and unblocking mechanism

- `sigprocmask` function

■ Supporting functions

- `sigemptyset` – Create empty set
- `sigfillset` – Add every signal number to set
- `sigaddset` – Add signal number to set
- `sigdelset` – Delete signal number from set

Temporarily Blocking Signals

```
sigset_t mask, prev_mask;

Sigemptyset(&mask);
Sigaddset(&mask, SIGINT);

/* Block SIGINT and save previous blocked set */
Sigprocmask(SIG_BLOCK, &mask, &prev_mask);

:   /* Code region that will not be interrupted by SIGINT */

/* Restore previous blocked set, unblocking SIGINT */
Sigprocmask(SIG_SETMASK, &prev_mask, NULL);
```