fork()

Objectives

- Explain how to create new processes under Unix
- Explain how `fork()` could be implemented under TOS

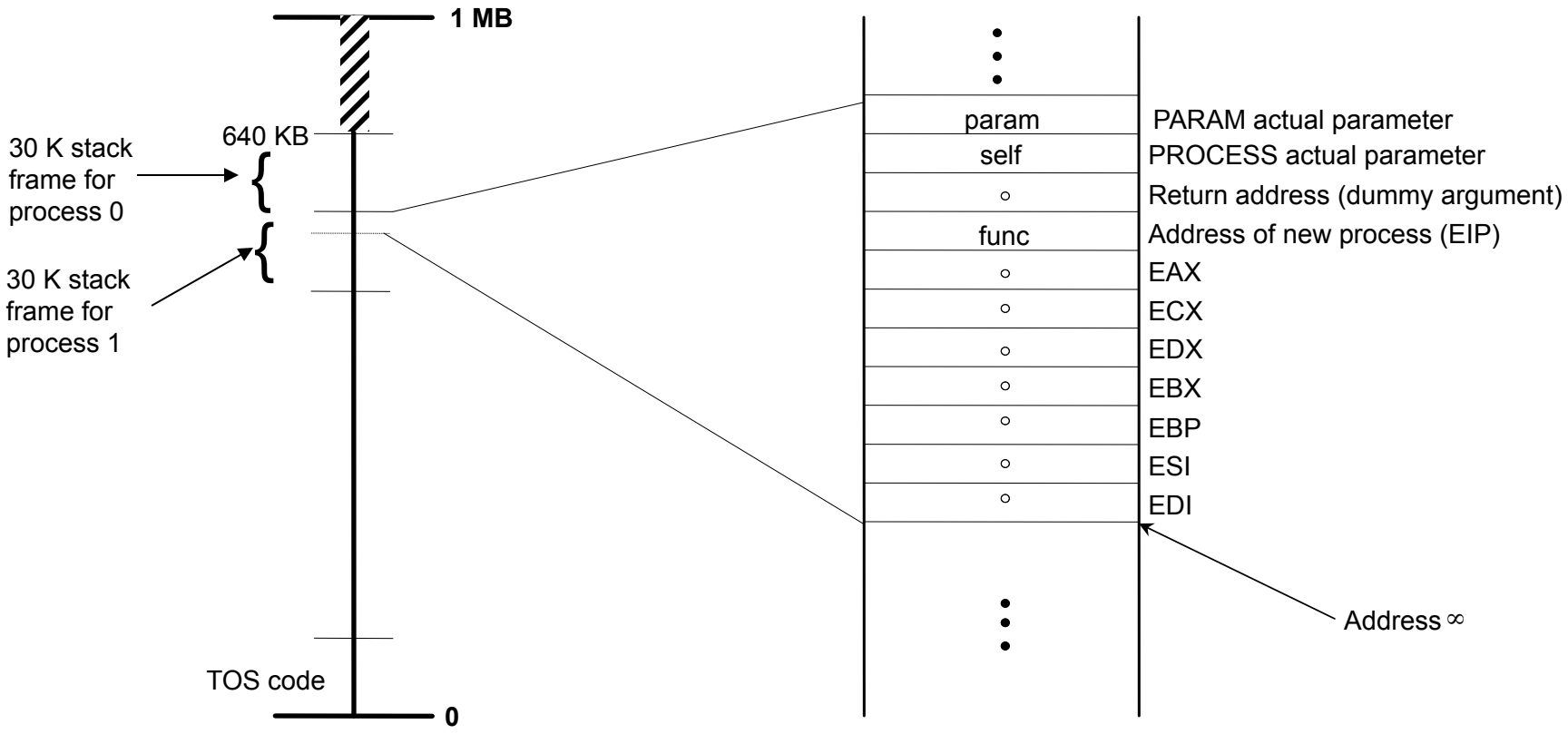
Review: create_process()

- New processes are created in TOS using `create_process()`
- `create_process()` does:
  - Allocate a free PCB entry
  - Initialize the PCB entry
  - Setup the initial stack frame
- The entry point of a TOS process is defined via a function pointer

Example: create_process()

```
void test_process (PROCESS self, PARAM param)
{
    // assert (self->name == “Test process”)
    // assert (param == 42)
}

void kernel_main ()
{
    // …
    create_process (test_process, 5, 42,
                    "Test process");
}
```

Stack of the new process



Address ∞ will be saved in PCB.esp

Overview of fork()

- In Unix, a new process is created via a call to `fork()`.
- `fork()` creates an exact copy of the calling process.
- Exact copy of:
  - Program code
  - Heap
  - Stack

fork() man page

NAME

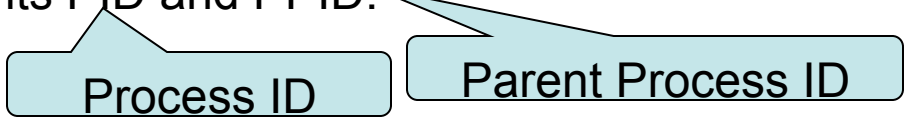
fork - create a child process

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
```

DESCRIPTION

fork creates a child process that differs from the parent process only in its PID and PPID.



RETURN VALUE

On success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution. On failure, a -1 will be returned in the parent's context, no child process will be created, and errno will be set appropriately.

fork() – Example under Unix

```
#include <iostream>
#include <unistd.h>

void main()
{
    int x = 42;

    if (fork() != 0)
       cout << "Parent process. x=" << x << endl;
    else
       cout << "Child process. x=" << x << endl;
}
```

Output:

```
Parent process. x=42
Child process. x=42
```

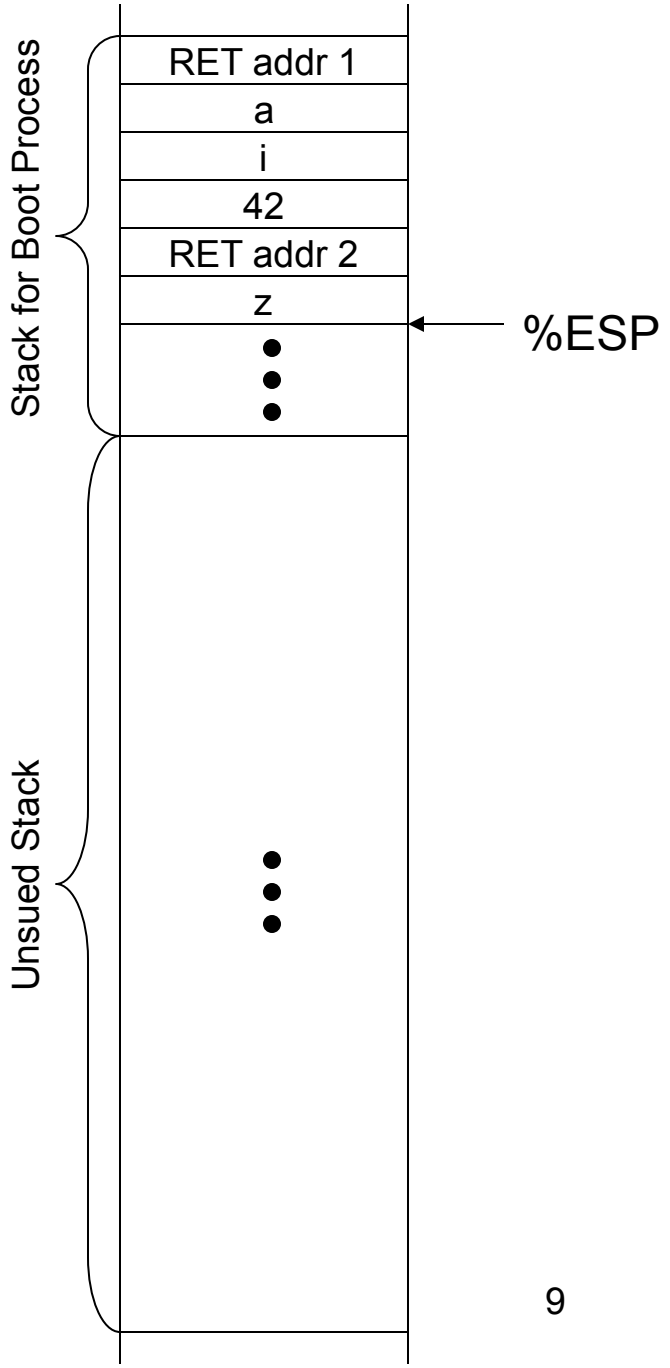
Before calling fork()

```
void f (int x)
{
    int z;
    fork();
}

void g()
{
    char a;
    int i;
    f (42);
}

void kernel_main ()
{
    g();
}
```

Before fork()

RET addr 2

RET addr 1



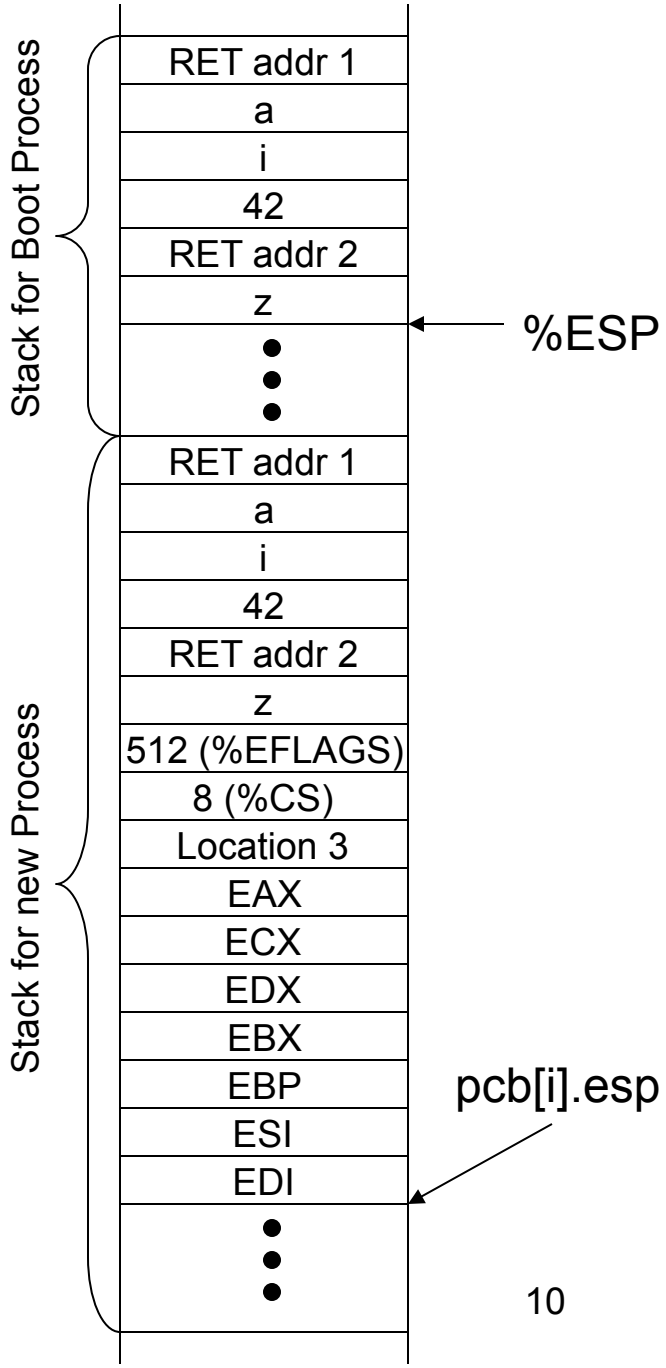
After calling fork()

```
void f (int x)
{
    int z;
    fork();
}

void g()
{
    char a;
    int i;
    f (42);
}

void kernel_main ()
{
    g();
}
```

Location 3

RET addr 2

RET addr 1



Loading Processes

- If `fork()` only creates a copy of the parent process, how can different programs be run under Unix?
- Solution: `execve()` loads a new program.
  - A call to `execve()` loads a new program into the running process.
  - A call to `execve()` therefore never returns (unless there is an error).

fork/wait/execve Example

```
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>

void main()
{
    int x = 42;

    if (fork() != 0) {
        int status;
        cout << "Parent process. x=" << x << endl;
        wait (&status);
    } else {
        cout << "Child process. x=" << x << endl;
        char* args[] =  {"ls", NULL};
        char* env[] =  {NULL};
        execve ("/bin/ls", args, env);
        cout << "Never reached" << endl;
    }
}
```