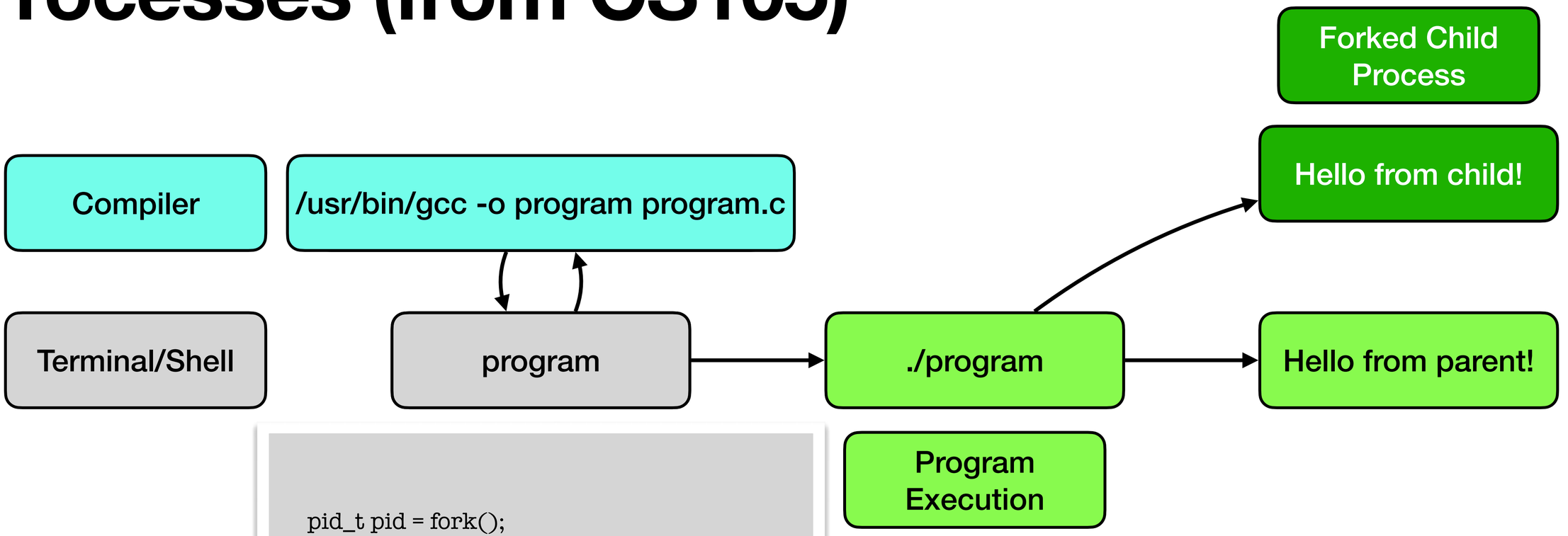
(Re-)Introducing Processes

Colloquium on Friday!
HW1 (Code Review) in class today!
HW2 Released!

Outline

- Processes and threads (the 105 perspective)
- Loading programs into the process structure
- Code Review
- The organization of processes, threads, and contexts

Processes (from CS105)




```
pid_t pid = fork();

if (pid == 0) {
   printf("Hello from child!\n");
} else {
   printf("Hello from parent!\n");
}
```

1 Name all of the possible outputs from the execution of ./program

2 How many processes depicted in this system?

Processes (from CS105)

- A *process* is an instance of any running program (including the shell!)
- Many processing platforms support multiprocessing (e.g., running different processes concurrently on different processors)
- Everything that is needed to be known to execute a process is stored in the *Process Control Block* (PCB)
- The PCB contains information concerning: executable, arguments, errno value, status, file table, register values, memory state, scheduling information

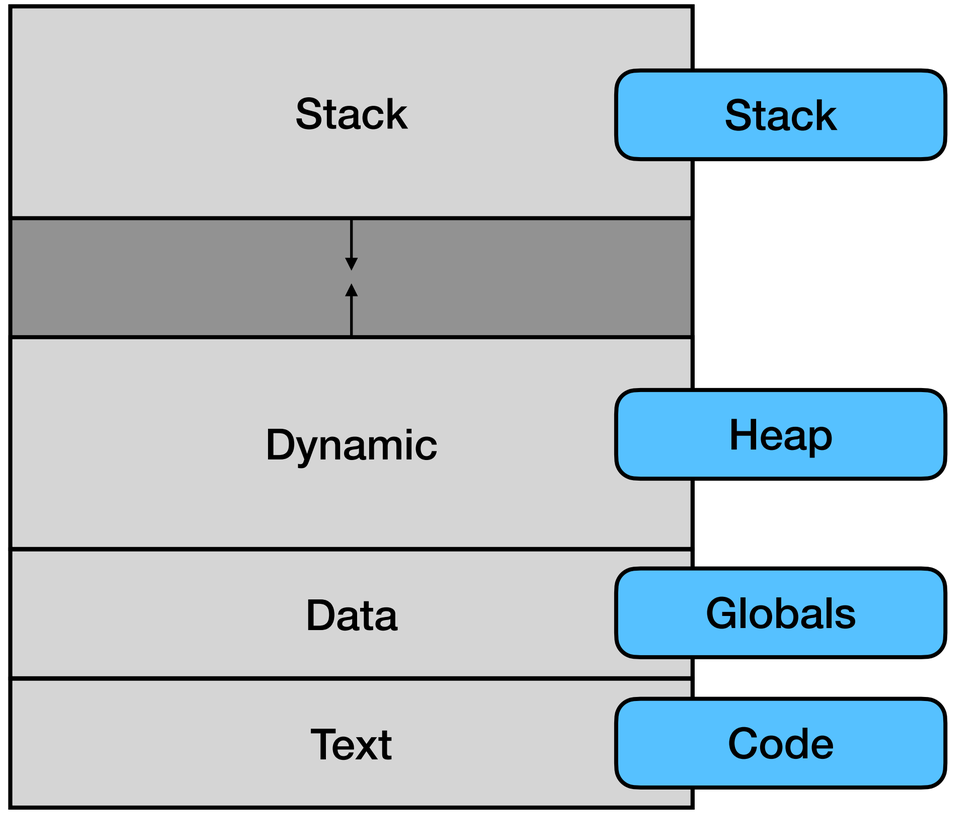
Process Address Spaces (from CS105)

```
pid_t pid = fork();

if (pid == 0) {
    printf("Hello from child!\n");
} else {
    printf("Hello from parent!\n");
}
```

program

In Linux, this process is performed by the *loader* ➡️ the OS kernel handler for the `execve` system call



Chat with your neighbor(s)!

Recall that the fork system call is called once, but returns twice (with two different processes). Thinking about the PCB below, what would be directly copied from a parent process into its child on a call to fork? What needs to be written anew?

```c
/* Process descriptor */
typedef struct proc
{
    pid_t p_pid;                    // Process ID
    char p_name[MAX_STRING_LEN]; // Process name

    list_t p_threads;               // Threads list
    spinlock_t p_threads_lock;   // lock for threads list
    list_t p_children;              // Children list
    spinlock_t p_children_lock;  // lock for children list
    struct proc *p_pproc;           // Parent process

    list_link_t p_list_link;     // Link of list of all processes
    list_link_t p_child_link;    // Link on parent's list of children

    long p_status;                  // Exit status
    proc_state_t p_state;           // Process state
} proc_t;
```

HW1: Code Review

- print effective userid
- update user's authentication tokens
- exit a login shell
- print name of current/working directory
- list directory contents
- change the current directory to dir; if dir is not supplied, the value of the HOME shell variable is the default
- make/remove directories
- copy files and directories
- move (rename) files
- remove files or directories
- change file mode bits
- concatenate files and print on the standard output
- opposite of more
- show who is logged on and what they are doing
- show a listing of last logged in users
- using pipes to combine functionality
- controlling process functionality

- 10 minutes per group
- Part 1: copy-editing
  - Consistent, readable style is sufficient for this class ☑️
- Part 2: test cases
  - (1) Describe the test case at a high-level
  - (2) Describe the inputs to the test case (you may want to show the code that you wrote and used)
  - (3) If the test case failed, describe the errors and what might be causing the errors (you can do this together!)
- What did you learn? Was there any syntax/tricks that you will takeaway? Did anything stand out?

HW1: Remaining Logistics

- Programming component is done ✅
- Code Review has been completed ✅
- Code Review Response due by Friday ☑️
- If you would like to submit changes before I look at your code (for thoughts and feedback), you can continue to do so through Friday ☑️
- Homework 2 released! Details on the course page (programming due Sept 28)

HW2 Partners

Group 1: Katherine + Anika

Group 2: Livia + Ishita

Group 3: Henok + Tung

Group 4: Tian + Vadym

Group 5: Alan + Yaseen

Group 6: Larry + Sudharsan

Group 7: Philippe, Jalen, Drew

Group 8: Chengyi, Sadhvi, Aly

Code Review Pairs

Group 7 + Group 6
Group 2 + Group 3
Group 5 + Group 4
Group 1 + Group 8

Processes, Threads, and Contexts

```c
/* Process descriptor */
typedef struct proc
{
    pid_t p_pid;                    // Process ID
    char p_name[MAX_STRING_LEN]; // Process name

    list_t p_threads;               // Threads list
    spinlock_t p_threads_lock;      // lock for threads list
    list_t p_children;              // Children list
    spinlock_t p_children_lock;     // lock for children lis
    struct proc *p_pproc;           // Parent process

    list_link_t p_list_link;        // Link of list of all p
    list_link_t p_child_link;       // Link on parent's list

    long p_status;                  // Exit status
    proc_state_t p_state;           // Process state
} proc_t;
```

```c
/* Thread descriptor */
typedef struct kthread {
    char *kt_kstack;            // kernel stack pointer
    void *kt_retval;            // return value
    long kt_errno;              // errno of most recent syscall
    context_t kt_ctx;           // thread context

    struct proc *kt_proc;       // corresponding process

    long kt_cancelled;          // set if the thread has been cancelled
    kthread_state_t kt_state; // thread state

    spinlock_t kt_lock;         // so thread state doesn't change during clone

    list_link_t kt_plink;       // link on process's thread list, p_threads
    list_link_t kt_qlink;       // link on ktqueue if thread isn't running
} kthread_t;
```

```c
typedef struct context {
    // general registers
    void *c_rax;
    void *c_rbx;
    void *c_rcx;
    void *c_rdx;

    // index and pointers
    void *c_rip;
    void *c_rsp;
    void *c_rbp;
    void *c_rsi;
    void *c_rdi;

    void *c_flags;

    page_table_t *page_table;

    char *c_kstack;
    unsigned long c_kstacksz;
} context_t;
```

Processes, Threads, and Contexts

```c
/*
 * Executes a thread function helper for setting up stack in new thread
 */
static void __context_thread_initial_func(context_func_t func, long arg1,
                                          void *arg2) {
    void *result = func(arg1, arg2);
    kthread_exit(result);
}

/*
 * Sets up a context to begin execution
 */
void context_setup(context_t *c, context_func_t func, long arg1, void *arg2,
                   void *kstack, int kstacksz, page_table_t *pt) {
    c->c_kstack = kstack;
    c->c_kstacksz = kstacksz;
    c->page_table = pt;

    // put initial arguments onto the stack and leave room for the return value
    // from context thread initial_func (which calls passed func)
    c->c_rsp = kstack + kstacksz;
    c->c_rdx = arg2;
    c->c_rsi = arg1;
    c->c_rdi = func;

    // set frame pointer to start of stack pointer
    c->c_rbp = c->c_rsp;
    c->c_rip = (void *) __context_thread_initial_func;
}
```

Basic setup of the stack and page table for the process

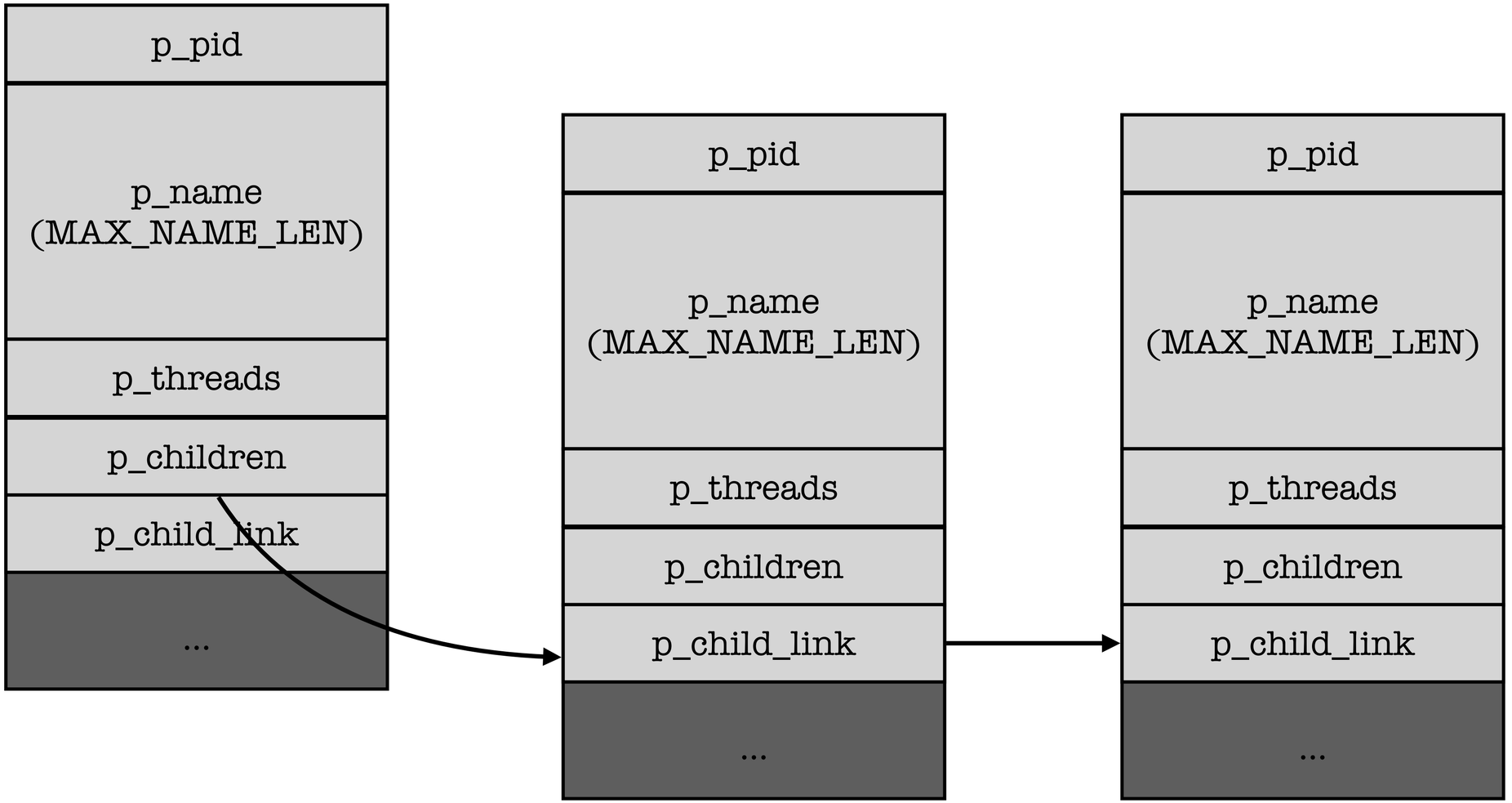
Setup initial register values, arg1 and arg2 are the same inputs as the parameters to `main`!

Execute the "__context_thread_initial_func" so that the thread can exit cleanly!

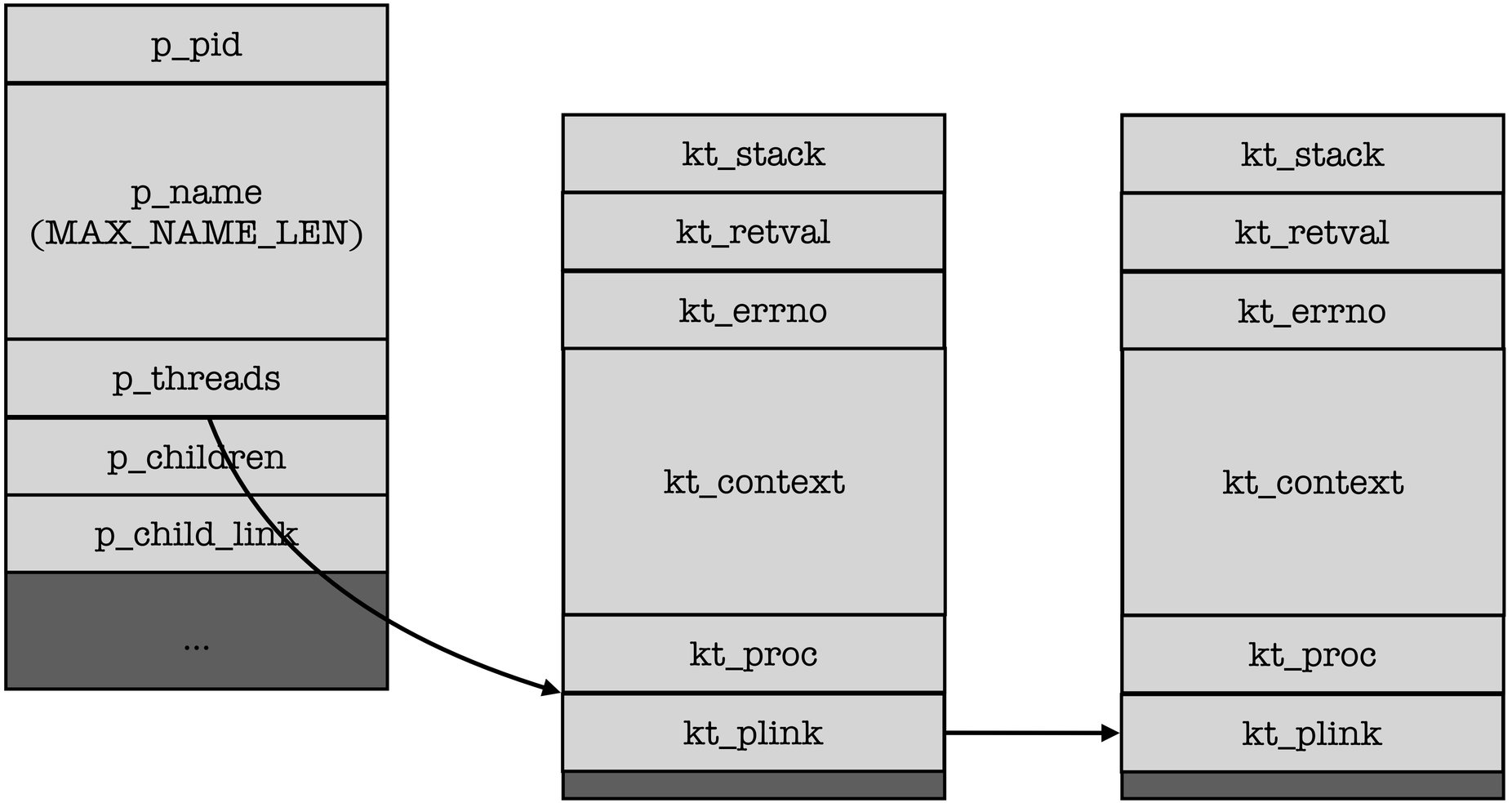
Processes, Threads, and Contexts

- To maintain a process control block (PCB), the processes subsystem of the operating system needs to define a process struct ➡️ this is the same with the thread control block (TCB) and its associated context
- When “loading a program” into a context, we set the appropriate fields with the “context” attribute of the TCB
  - The instruction pointer should point to the beginning of how the operating system wants to start executing the program with a clean exit (e.g., `kthread_exit`)
  - The register values/stack should be set to the appropriate fields the ABI defines to run the function

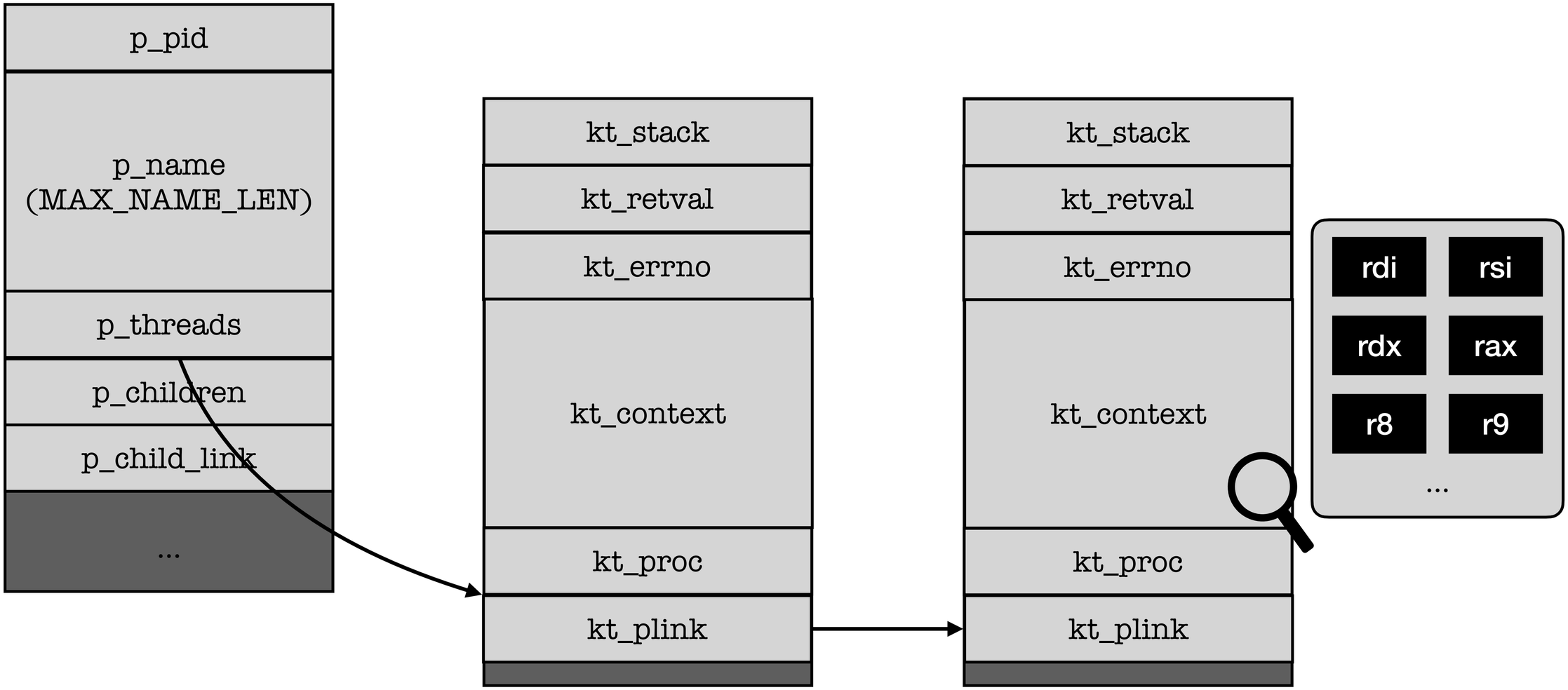
Managing Processes



Managing Processes (+ threads)



Managing Processes (+ threads (+contexts))



Summary

- As presented in CS105, processes are one of the most important abstractions that we have in operating systems as it provides the majority of mechanisms for the OS to act as a referee, illusionist, and glue
- To “load a program” into a process (the function of the `exec` system call and the OS loader), the OS needs to set the thread context according to the ABI to run a program
- When managing these concepts, everything needs to be organized in OS data structures where attributes may communicate with one another to implement the functionality