Computer Systems

Exercise 2

Process

- Instance of a program
- Ingredients of a process:
  - Virtual processor (address space, registers, program counter, instruction pointer)
  - Program text (code)
  - Program data (heap, stack)
  - OS stuff (open files, sockets, CPU shares, security rights)
- Executed in an “execution environment” (virtual address space, available system calls, etc.)
- Purpose of process concept
  - Protection of program data
  - Running of program
- Identified by PID (process ID)
- Software = running processes + kernel

Process creation – spawning new process

Did it work?

```
BOOL CreateProcess(
                                             address of binary
  in_opt       LPCTSTR              ApplicationName,
  inout_opt    LPTSTR               CommandLine,
  in_opt       LPSECURITY_ATTRIBUTES ProcessAttributes,
                                       e.g. rights
  in_opt       LPSECURITY_ATTRIBUTES ThreadAttributes,
  in           BOOL                 InheritHandles,
                                    of the calling process (e.g open files)
  in           DWORD                CreationFlags,
  in_opt       LPVOID               Environment,
  in_opt       LPCTSTR              CurrentDirectory,
  in           LPSTARTUPINFO        StartupInfo,
  out          LPPROCESS_INFORMATION ProcessInformation
                                       name, ID
);
```

What to run?

What rights will it have?

What will it see when it starts up?

The result

Moral: the parameter space is large!

Process creation – fork() and exec()

- Simplifies creating processes:
  - Fork creates a copy of the current process, same only that child has return value of fork() = 0 and parent has return value of fork() = PID of child
  - Exec() replaces text of calling process with new program.

→ Creates tree of processes

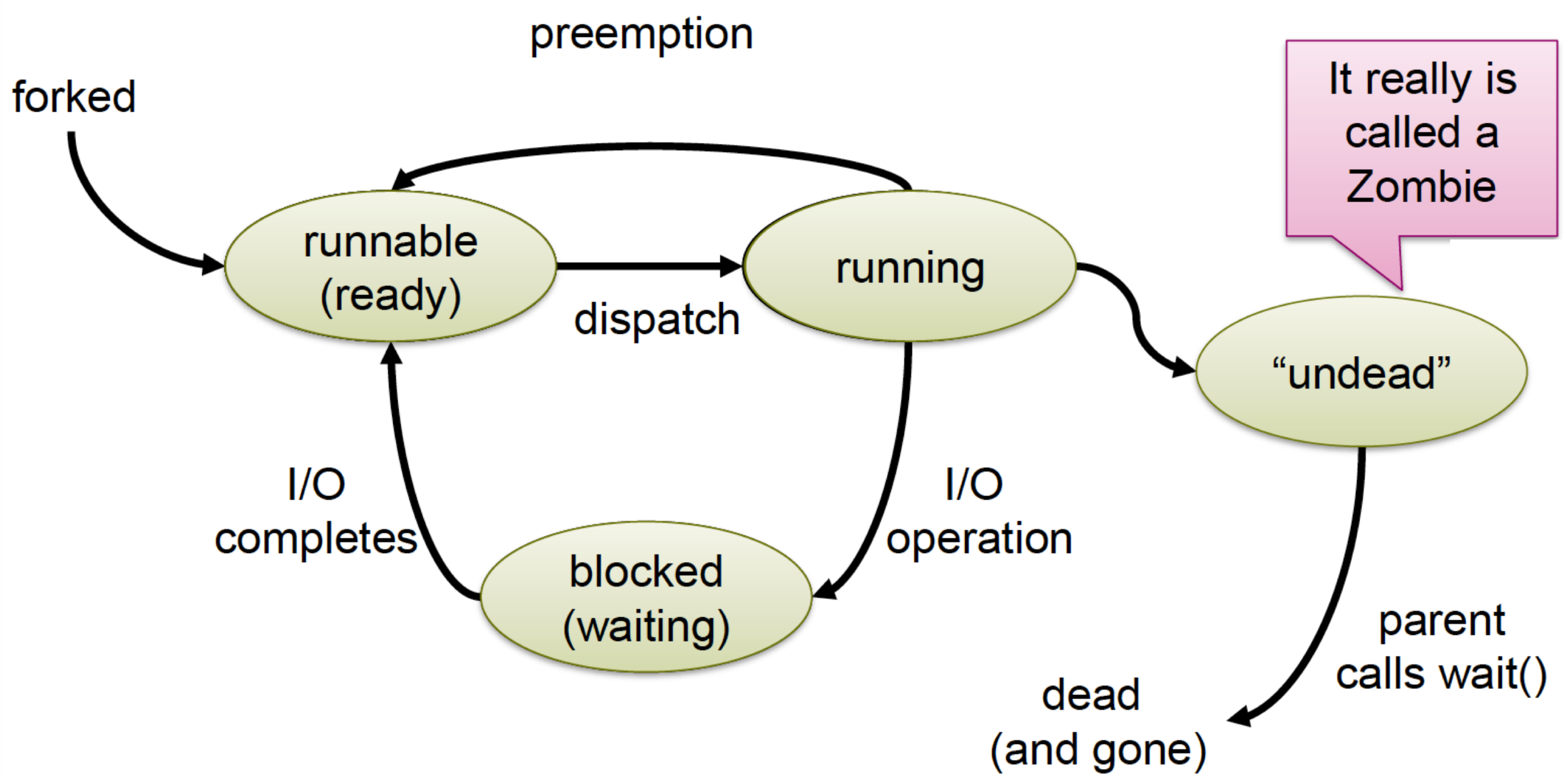
Example fork(), exec()

```
pid_t p = fork();
if ( p < 0 ) {
   // Error…
   exit(-1);
} else if ( p == 0 ) {
   // We're in the child
   execlp("/bin/ls", "ls", NULL);
} else {
   // We're a parent.
   // p is the pid of the child
   wait(NULL);
   exit(0);
}
```

Return code from fork() tells you whether you're in the parent or child (cf. setjmp())

Child process can't actually be cleaned up until parent "waits" for it.

Process lifecycle



Zombies & Orphans

- Why Zombies?
  - If no Zombies, child process could fail and nobody would know because parent has no chance to catch exit status
- What happens with child process whose parents have exited?
  - They are called orphans and get “adopted” by the init process (PID = 1)

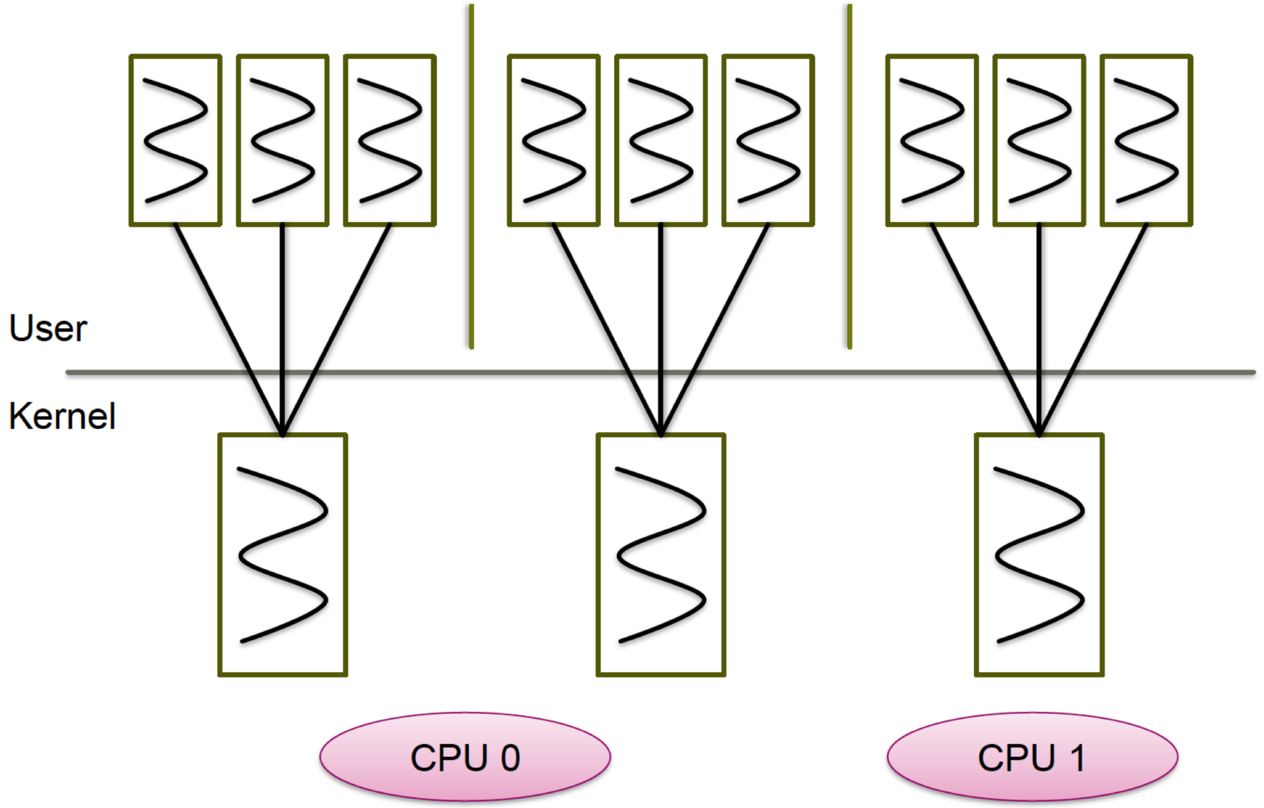
Coroutines

- Concurrency without parallelism
- Basically two functions that call each other (symmetric relationship)

Threads

- A thread is part of a process. Multiple threads can exist in one process and share resources such as memory.
- Process – unit of resources, thread – unit of scheduling/execution
- User threads (lightweight processes)
  - Kernel is unaware of them, scheduled in user space
  - Fast to create and manage, but can’t make use of multithreading
- Kernel threads
  - At least on kernel thread exists for each process
  - One kernel thread can be mapped to each logical core and can be swapped once it gets blocked, but takes long to swap

User vs. Kernel Threads

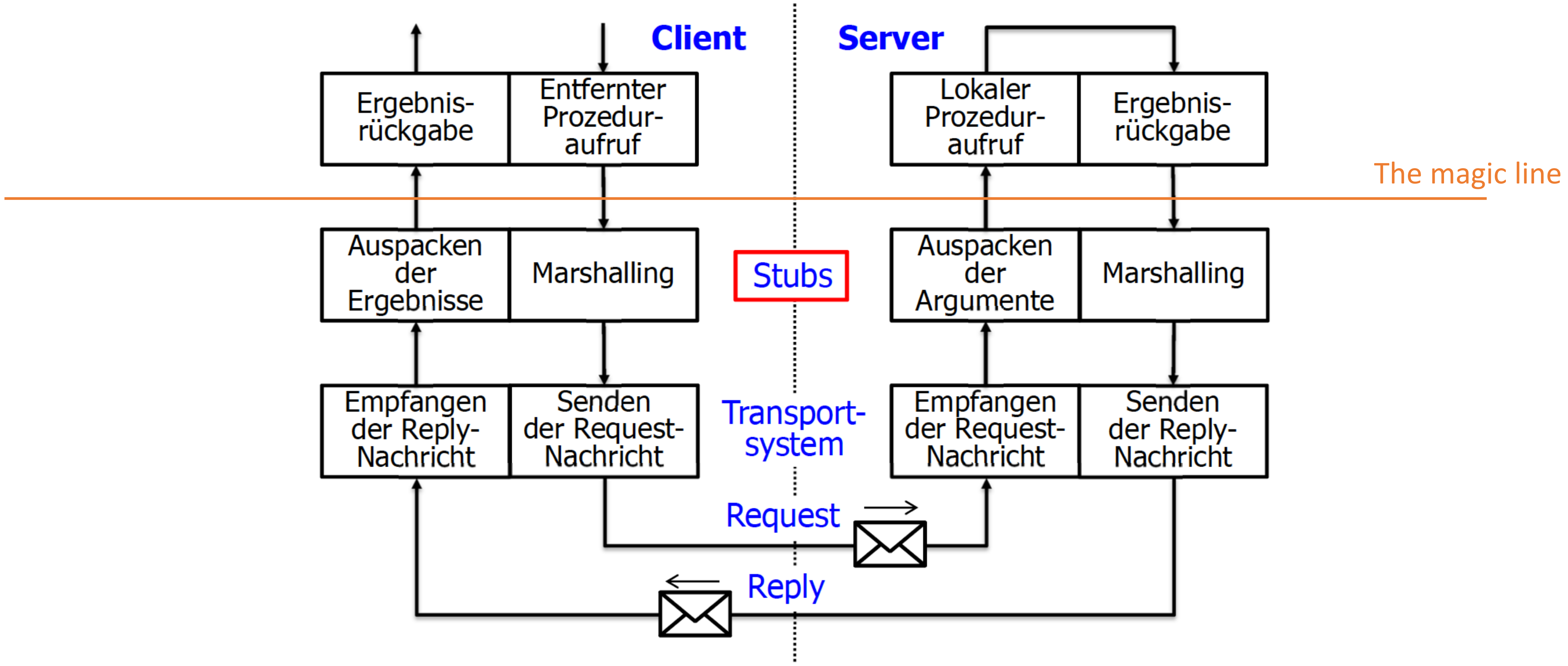


How do processes communicate?

- **Shared Memory**
  - Semaphores/Locks
    - Synchronization instructions (CAS, TAS, LL/SC)
  - Transactional Memory

- **Message passing**
  - Asynchronous/Synchronous
  - Blocking/non-blocking
  - Pipes
    - Named/unnamed
  - Upcalls/Signals
    - SIGSEGV from memory management
    - SIGKILL from other process
  - RPC

RPC



Client
Server
Ergebnis-rückgabe
Entfernter Prozedur-aufruf
Lokaler Prozedur-aufruf
Ergebnis-rückgabe
The magic line
Auspacken der Ergebnisse
Marshalling
Stubs
Auspacken der Argumente
Marshalling
Empfangen der Reply-Nachricht
Senden der Request-Nachricht
Transport-system
Empfangen der Request-Nachricht
Senden der Reply-Nachricht
Request
Reply

Quiz

- **What is the relation between a process and a program?**
  - A process is a running instance of a program
- **How are processes identified?**
  - By the process ID (PID)
- **In what state is a process after it exited?**
  - Zombie state
- **Are user threads or kernel threads easier to switch?**
  - User space threads
- **How long should you spin for waiting for a lock?**
  - One context switch time
- **What is the point of a name server?**
  - to hold interface references for services