Lecture 3

Processes

&

Threads

26 August, 2011

what is a **process**?

resource + execution

typical content of a **process control block**:

registers, state, priority, pid, parent
program counters, program status word
CPU time used
pointers to memory segments
working directory
opened files
user ID, group ID

etc.

OS maintains

process table

(one PCB / process)

A **CPU scheduler** decides which process to run

OS saves and restores PCBs to **context switch** between processes

when to context switch?

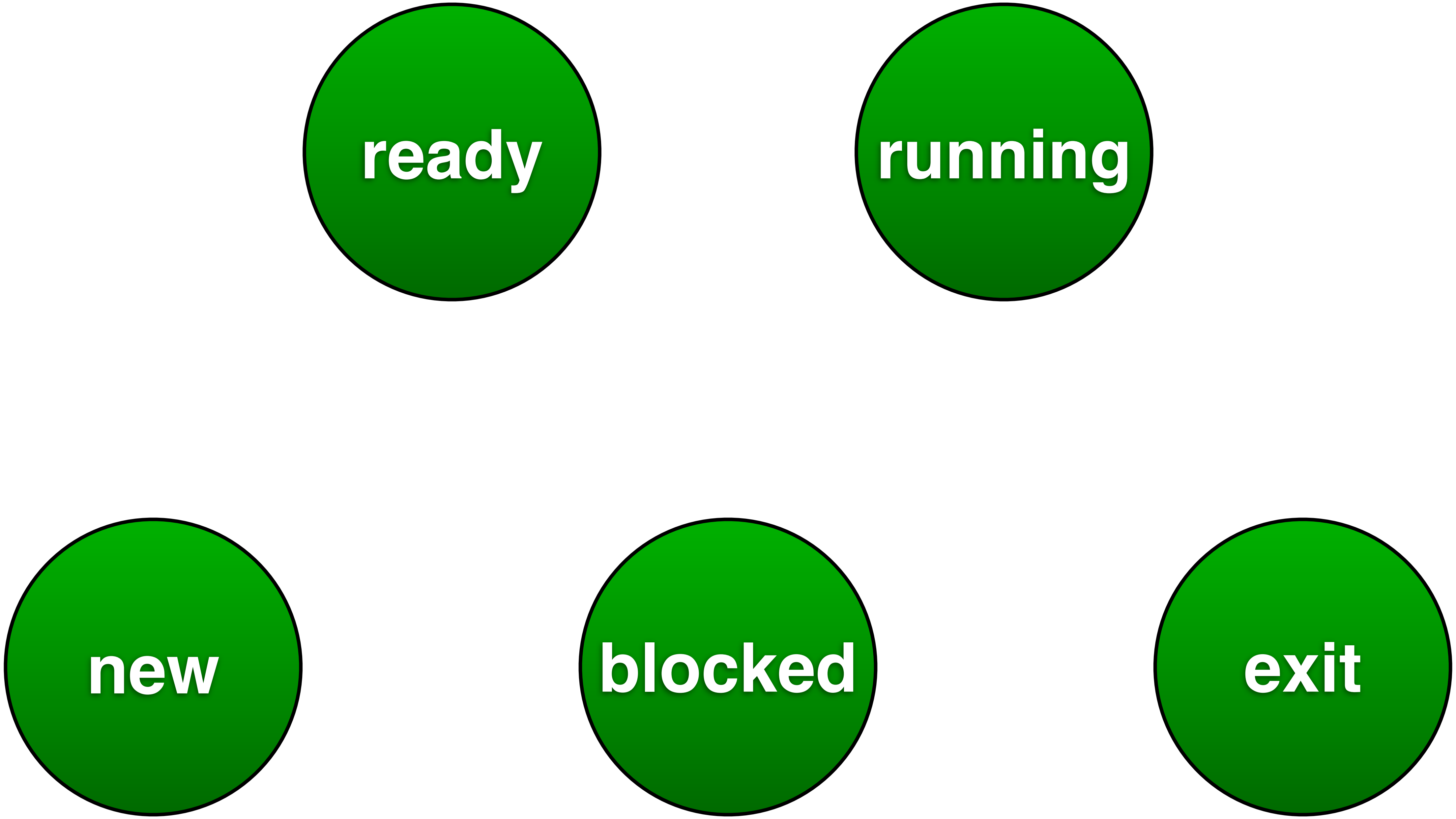
blocked I/O

e.g. Java InputStream’s read( )

interrupt

(system call, timer, I/O)

e.g. time allocated to a process is used up,
data ready to be read



ready

running

new

blocked

exit

which ready process to run next ?

what causes a new process to be created?

explicit creation through system call

system initialization

(e.g, Linux init process)

upon user requests

(e.g., double click an icon, typing a command)

what causes a process
to terminate ?

finish running

(with or without error)

fatal error

(an example from Lab 1)

killed by another process

system calls for process management

```
BOOL WINAPI CreateProcess(
  __in_opt     LPCTSTR lpApplicationName,
  __inout_opt  LPTSTR lpCommandLine,
  __in_opt     LPSECURITY_ATTRIBUTES lpProcessAttributes,
  __in_opt     LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in         BOOL bInheritHandles,
  __in         DWORD dwCreationFlags,
  __in_opt     LPVOID lpEnvironment,
  __in_opt     LPCTSTR lpCurrentDirectory,
  __in         LPSTARTUPINFO lpStartupInfo,
  __out        LPPROCESS_INFORMATION lpProcessInformation
);
```

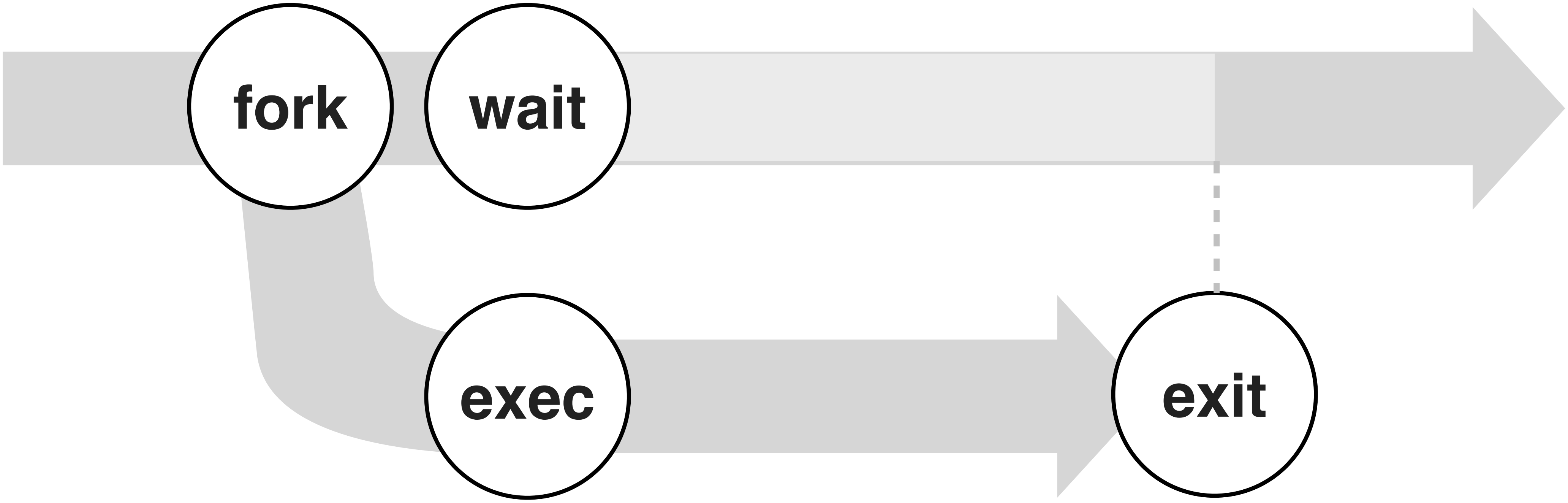
```
pid_t fork();
```

process hierarchy

POSIX standard

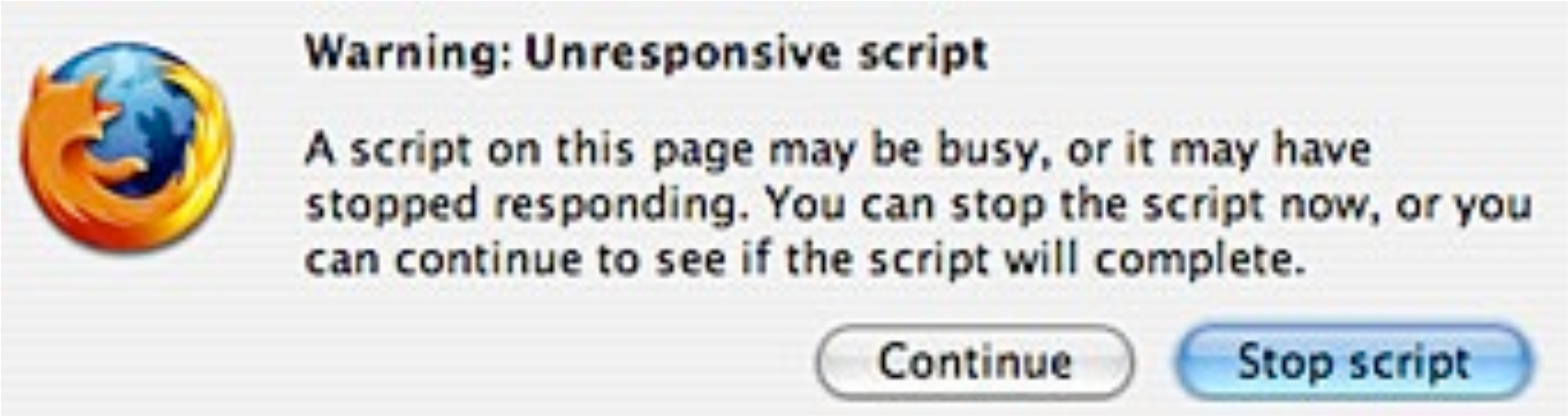
(Portable Operating System Interface for Unix) process-related system calls

fork, exec, wait, exit


fork
wait
exec
exit

zombie process
orphan process

consider a
Web browser



Warning: Unresponsive script

A script on this page may be busy, or it may have stopped responding. You can stop the script now, or you can continue to see if the script will complete.

consider a
Web server

concurrent multi-process server

```
**while** (1)
  block until new connection
  fork( )
  **if** (is child process)
    handle new connection
    exit( )
```

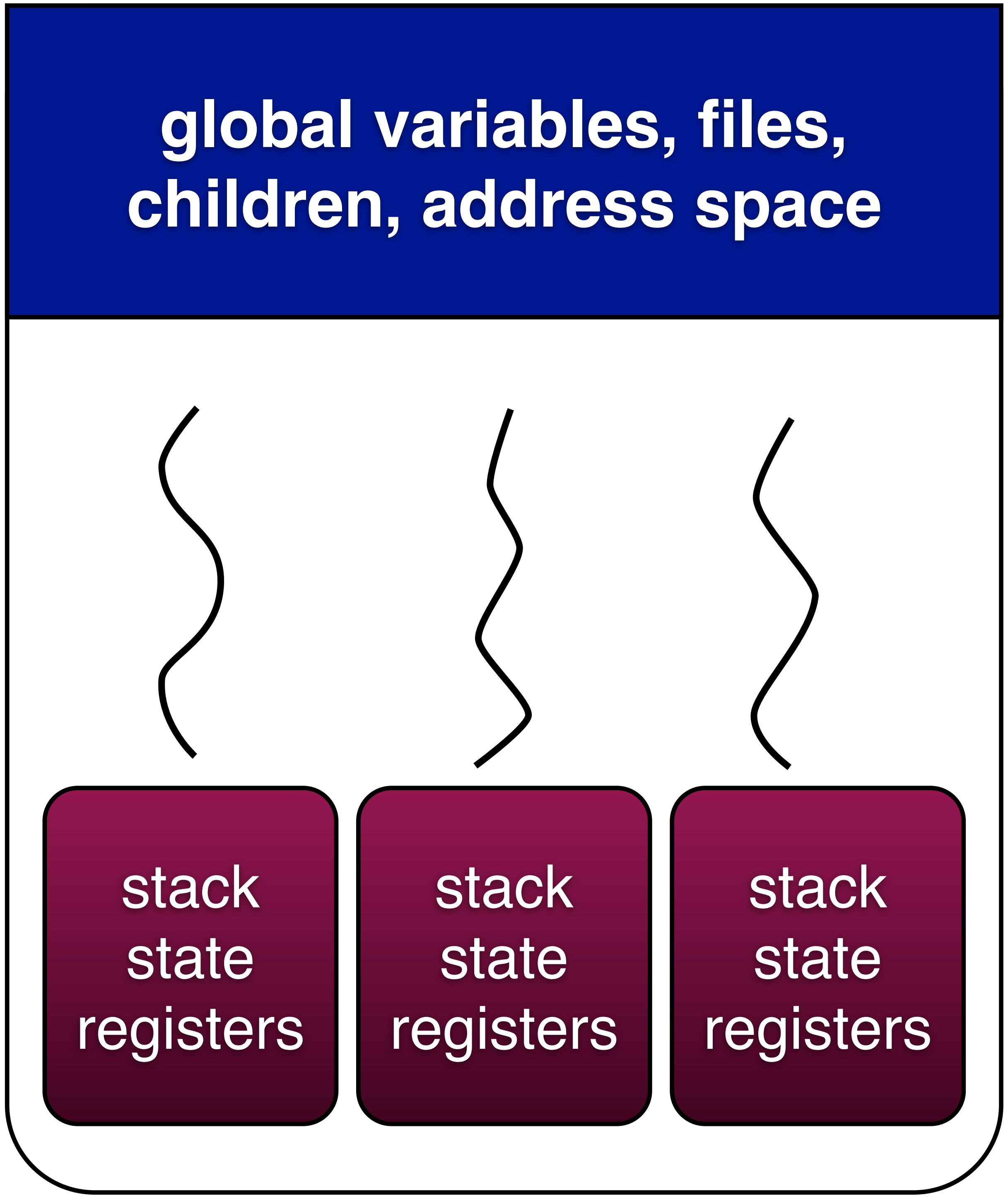
fork( ) is expensive

(do we really need to duplicate all the resources ?)

threads

same resource, different executions

a multi-threaded process



global variables, files, children, address space

stack
state
registers

stack
state
registers

stack
state
registers

advantages of multi-threading

vs single-threaded

improved responsiveness

exploits parallelism

abstraction for “independent” sequence of execution

vs multi-process

cheaper

allows sharing of resources

POSIX Threads API

```
pthread_create( )
pthread_exit( )
pthread_join( )
pthread_yield( )
```

thread scheduling
done by either
process or kernel

mixing threads and fork() can be tricky