

- multiprogramming
 - one process at a time
- multiprocessing
 - many processes at a time
- process
 - execution environment
 - provided by operating system
 - communicate with OS and with each other
 - system processes
 - user processes

- library files
 - collection of functions
 - available to programs
 - library function
 - runtime library functions
- static libraries
 - during linking
- shared object libraries
 - at load time
 - shared by many processes

● library files

- .a
- static
- ar utility
- examine library functions
- create library functions

```
linux$ g++ -c change_case.c
linux$ g++ -c ascii.c
linux$ ar cr libmy_demo.a ascii.o change_case.o
```

```
FILE: main.c
#include <iostream>
#include "my_demo.h"
...
```

```
linux$ g++ -o main main.c -L. -lmy_demo
```

```
-----+
FILE: ascii.c
char *
ascii( int start, int finish ) {
char *b = new char(finish-start+1);
for ( int i=start; i <= finish; ++i)
    b( i-start ) = char ( i );
return b;
}

FILE: change_case.c
#include <ctype.h>
char *
change_case ( char *s ) {
char *t = &s[0];
while ( *t ) {
    if ( isalpha(*t) )
        *t += islower(*t) ? -32 : 32;
    ++t;
}
return s;
}

FILE: my_demo.h
/*
prototypes for my_demo library functions
*/
#ifndef MY_DEMO_H

#define MY_DEMO_H
char * ascii ( int, int );
char * change_case ( char * );

#endif
-----+
```

- **system calls**

- request service from operating system
- OS performs work on behalf of caller
 - kernel
 - user mode
 - subset of instructions
- system (privileged) mode
 - entire instruction set

```
#apropos '(2'
```

- object code
 - library files and compiled code
 - combined at compile time
 - files not in standard library
 - specified at compile time

```
linux$ gcc prgm.c -lm
```

- include math library libm.a

- system calls / library function return codes
 - external global errno
 - defined constants in <sys/errno.h>
 - always examine return codes!!!!
 - **perror**
 - to produce error message

```
void perror ( const char *s );
```

- review **strerror**

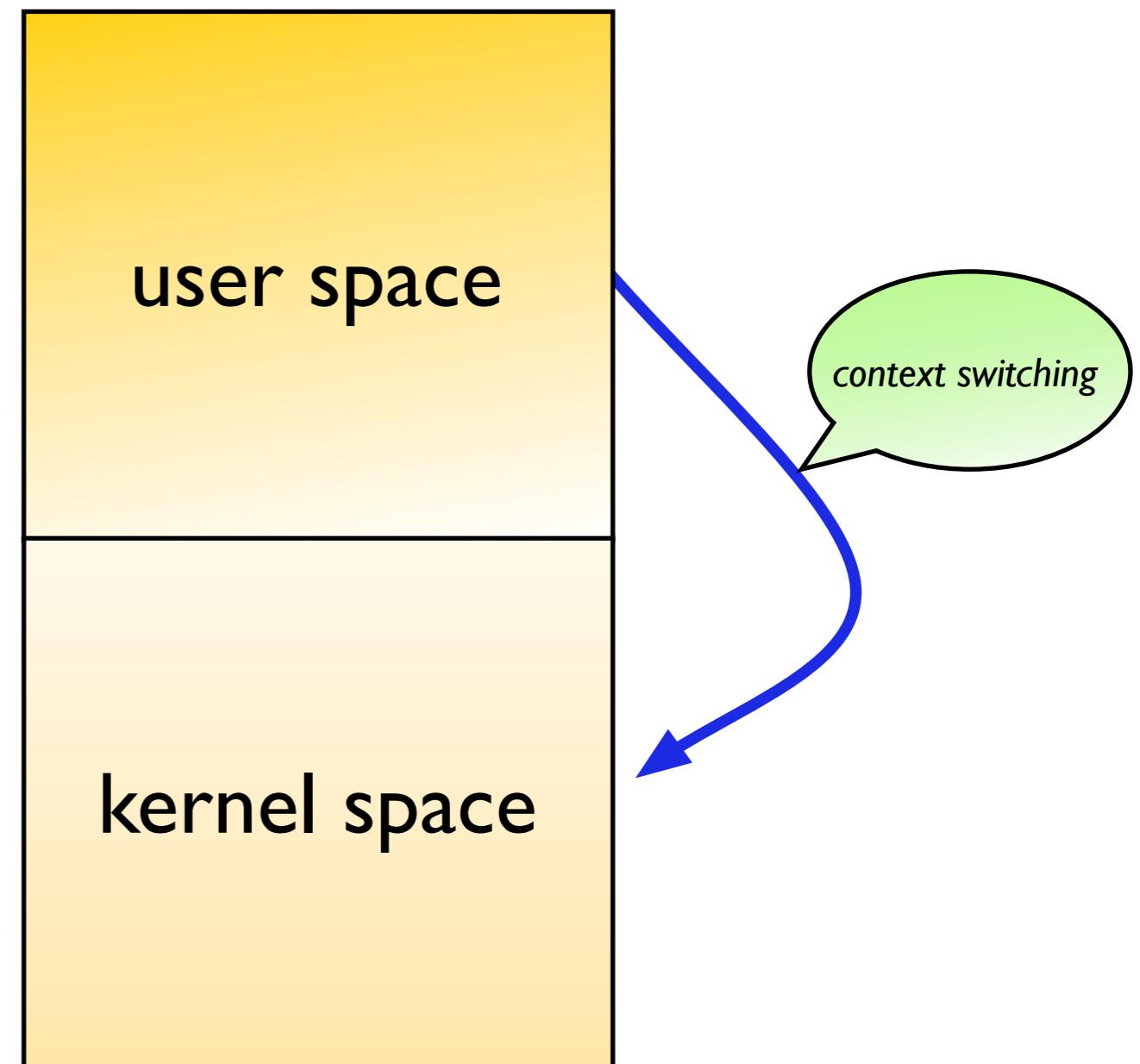
- ELF (executable and linking format)

- marked as executable
 - program text
 - data
 - relocation information
 - symbol table
 - string table

old format: a.out
Assembler OUtpuT Format

• System memory

- user space
 - user processes
 - protected
 - run in user mode
- kernel space
 - kernel execution
 - system processes
 - privileged mode



- process memory
 - private memory
 - divided into three segments
 - text
 - code
 - data
 - initialized variables
 - uninitialized variables
 - stack
 - automatic identifiers
 - function call information

- **text segment**
 - read-only segment
 - instruction segment
 - constant data
 - can be shared among processes
 - different instantiations of same code

- **data segment**
 - virtually contiguous to text segment
 - initialized/uninitialized data
 - may be expanded at execution time
 - new, malloc, calloc
- **stack segment**
 - automatic identifiers
 - register variables
 - function call information
 - grows towards uninitialized data segment

- u-area

- maintained by operating system
 - one per process
- specific process information
 - open files
 - current directory
 - signal actions
 - accounting information
 - a system stack segment
 - system calls
 - process can access information
 - via system calls

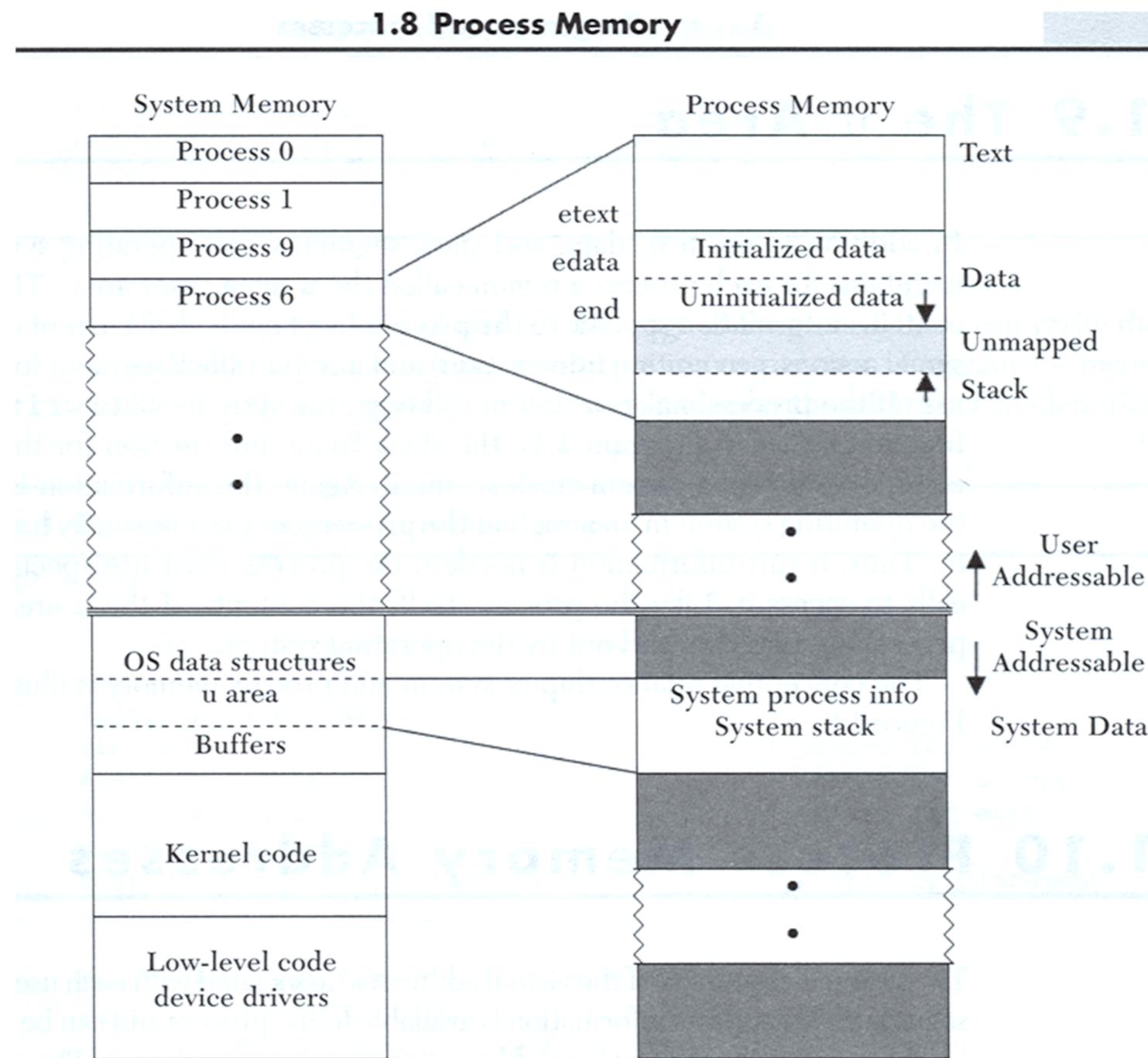


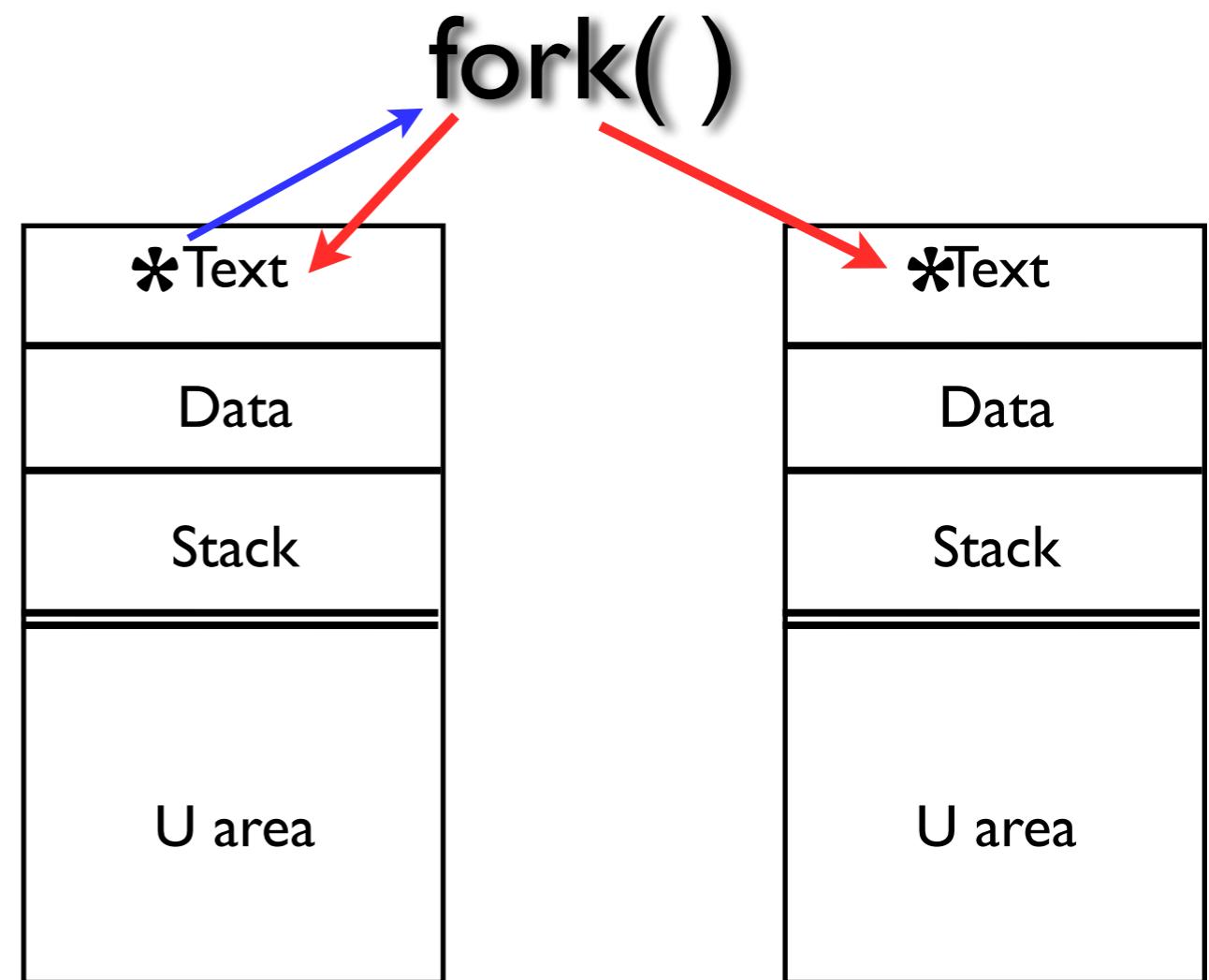
Figure 1.9 System and process memory.

- **fork ()**
 - parent process
 - process executing the fork()
- child process
- created by fork() system call

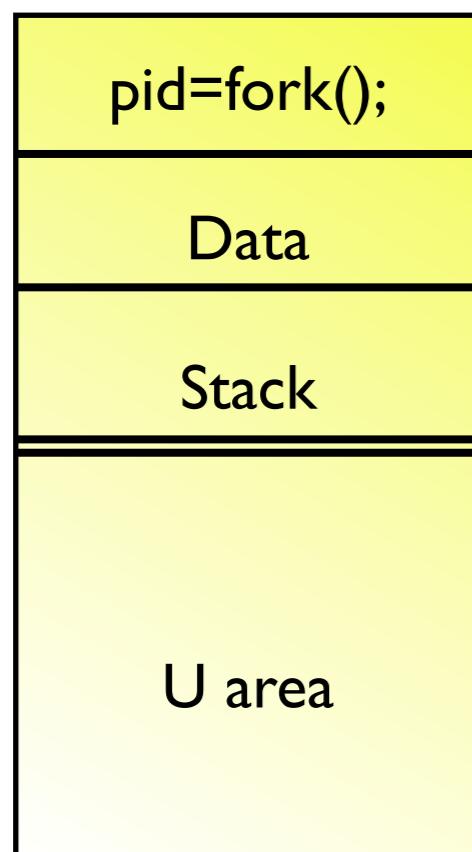
```
pid_t fork ( void );
```

Constant	perror Message	Explanation
EAGAIN	Resource temporarily unavailable	No memory to copy parents page table and allocate task structure
ENOMEM	Cannot allocate memory	Insufficient swap space

- returns twice
 - in parent
 - returns child PID
 - in child
 - returns 0
- process can identify itself
 - parent or child

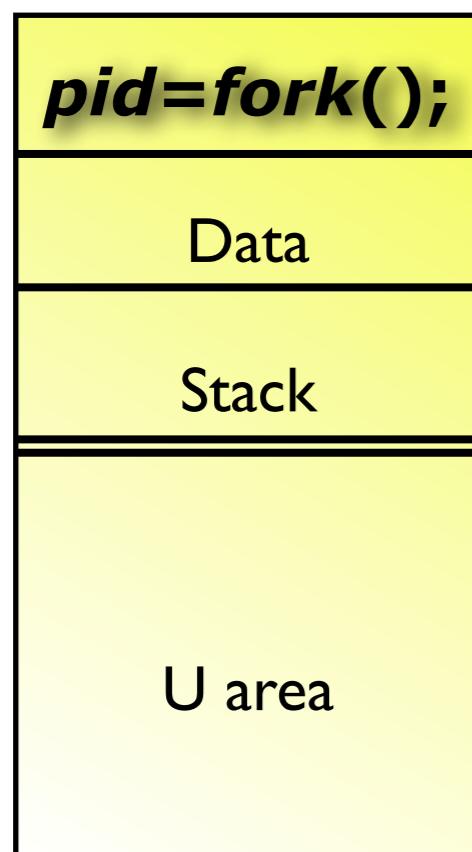


process id=789



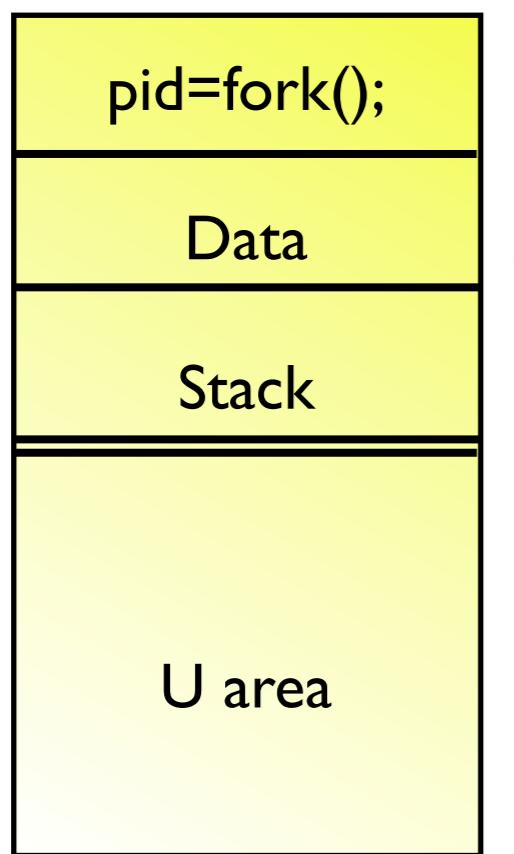
process executes a fork():

process id=789

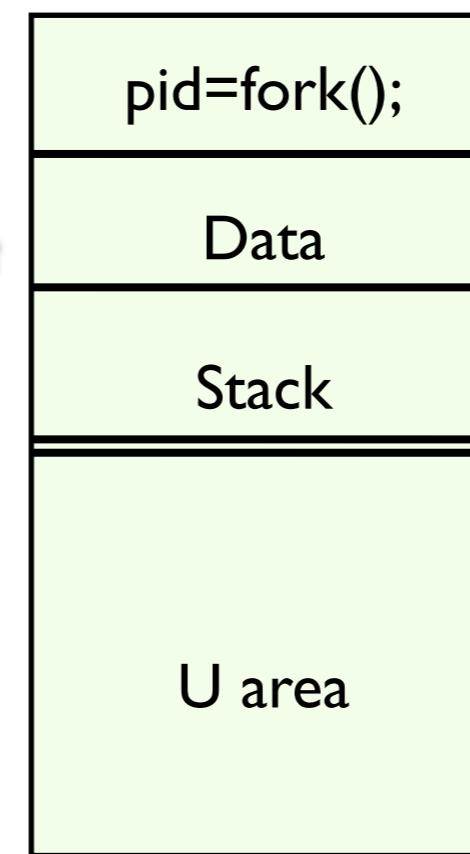
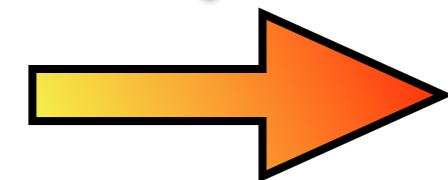


fork system call

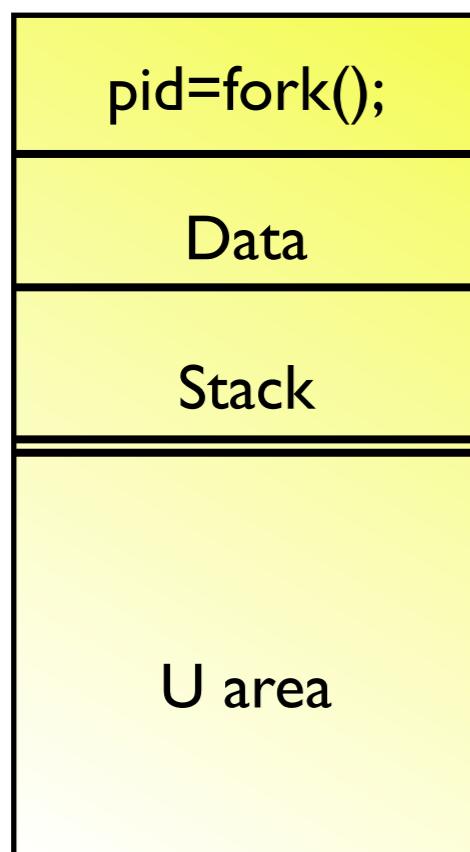
process id=789



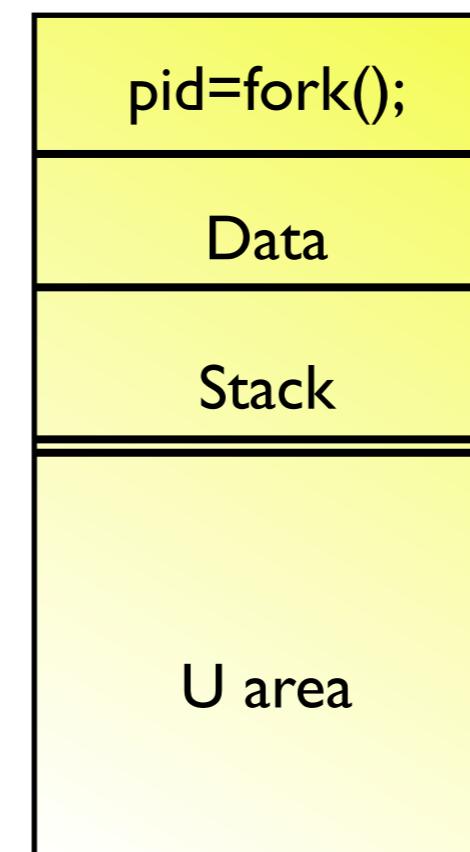
create process



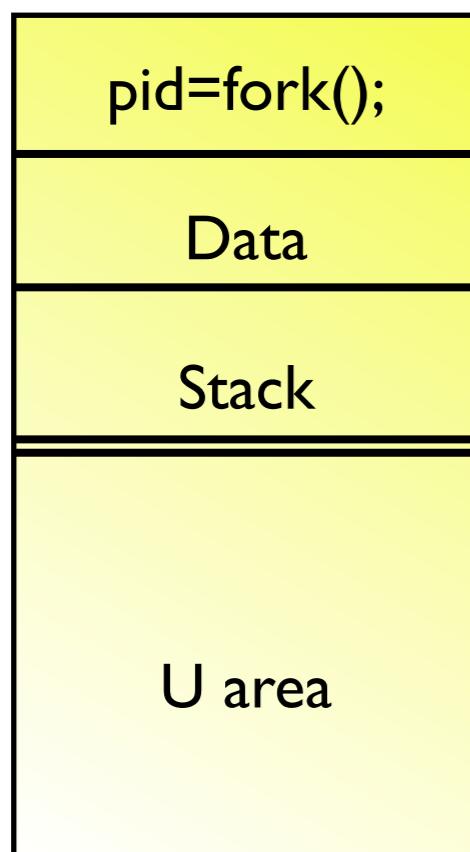
process id=789



process id=1089

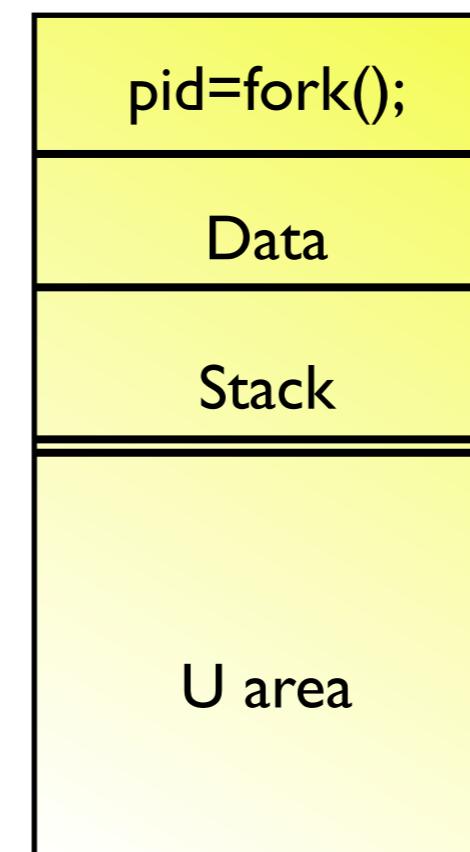


process id=789



pid = 1089

process id=1089



pid = 0

```
pid = fork ();
if ( pid == 0 ){
    // in child process
}
// in parent process
```

- Unique to process

- PID
- PPID
- System imposed process limits are reset
 - CPU allocation time
- File record locks are reset
- Signal actions

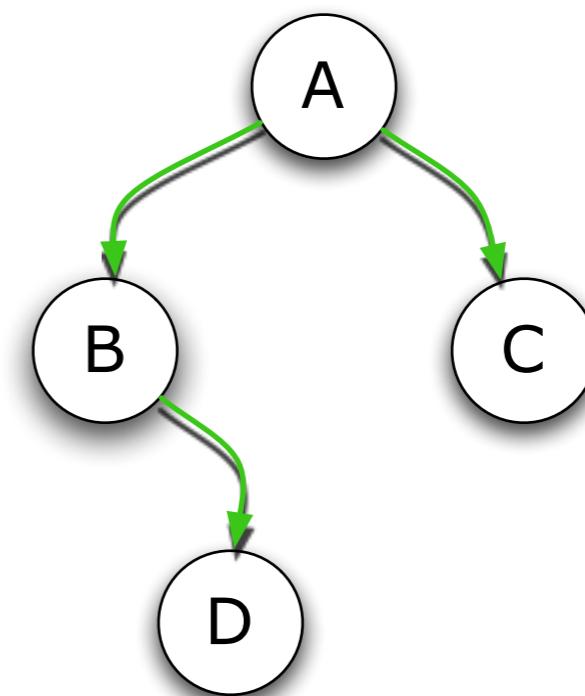
● Share

- real UID
- real GID
- effective UID
- effective GID
- process GUI
- controlling terminal
- set-user-id flag and set-group-id flag
- current working directory
- root directory
- file mode creation mask
- signal mask and dispositions
- the close-on-exec flag for any open file descriptors
- attached shared memory segments
- open file descriptors
- Environment information

- processes may create other processes

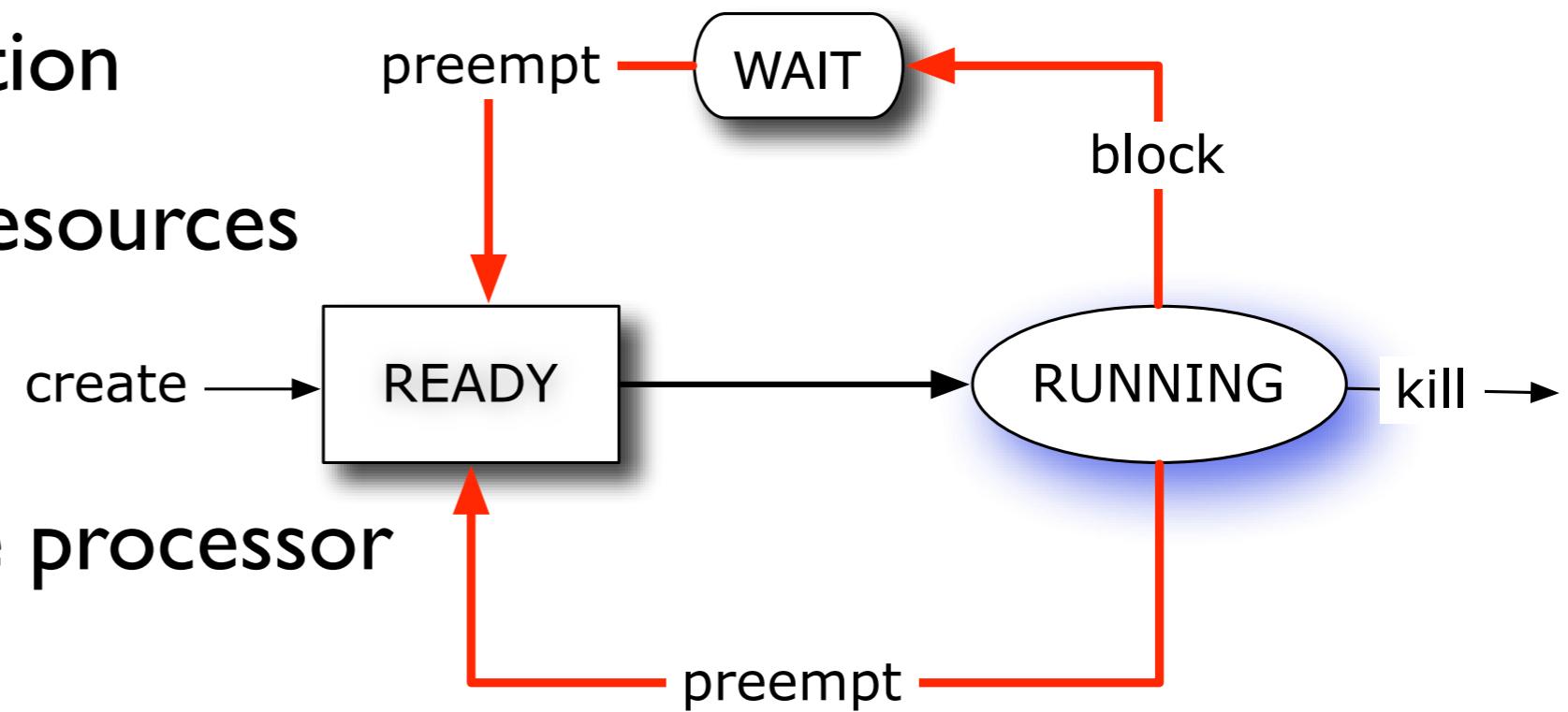
- parent - child relation

- A parent of B and C
- B parent of D



● Process States

- Ready
 - ready for execution
 - has all needed resources
- Running
 - in control of the processor
- Wait
 - waiting for an event to occur
 - waiting for a resource



- Process Control Block (PCB)
 - data structure created by the Operating System
 - one entry per process
 - create a process  entry in PCB
 - all information about a process
 - process id (PID)
 - dispatching information
 - priority
 - CPU time
 - resources
 - allocated
 - outstanding

A process is known through its PCB