Announcements

- another quiz/poll this morning
- posting sample solution to Lab01
- UNIX bootcamp this coming weekend
- CSSS election
  - more information at http://csss.usask.ca

Quote of the Day

- If the designers of X-Windows built cars, there would be no fewer than five steering wheels hidden about the cockpit, none of which followed the same principles -- but you'd be able to shift gears with your car stereo. Useful feature, that.
  - Marcus J. Ranum

UNIX File System Fundamentals

Noteworthy Directories

`/`	Root directory for the entire file system
`./`	Current directory
`~/`	Your home (login) directory
`~user/`	Home directory of given user
`..`	Parent directory

Noteworthy Directories

- `./` used to signify executing the named file in the current working directory as a command

  example: `./mycmd`

typical way you will execute your command "./a.out"
it will just invoke which ever file

File Conventions

- file “extensions” (c.f. Windows) are not required, but are useful
- for convenience don’t use spaces in filenames; instead try
  - `my_file_name`
  - `myFileName`

Basic File-Oriented Commands

- `cd`
- `pwd`
- `ls`
- `cp`
- `rm`
- `mv`
- `mkdir`, `rmdir`

File Permissions

- Files have three basic permissions
  - Read (r)
  - Write (w) **(edit)**
  - Execute/search (x)
- and three permission categories or levels of ownership
  - User (u)
  - Group (g)
  - Global/other (o)

File Permissions

- Controlled by a bit mask of the symbolic form
  - `rwxrwxrwx`
- each group of 3 bits corresponds to an ownership level
  - user group other

    ```
    rwx   rwx    rwx
    ```

    stored as an octal number from 0-7
    'rwx' = 111, 'r—' = 100 then change to octal
  - `r`, `w`, or `x` indicates permission on (a 1-bit)
  - – indicates permission off (a 0-bit)
- e.g. `ls -l temp`

File Permissions

- `chmod` **"change permissions on a file"**
  - to change file permissions
  - can use `chmod` symbolic-mode
    - e.g. `chmod g+w temp` **"turn on write for group"**

      ```
      chmod o-rwx temp
      chmod a+r temp
      chmod a+rwx temp
      ```
  - can use `chmod mask` where `mask` is a bit pattern in octal
    - e.g. `chmod 764 temp`

u - owner/user
g - group
o - other
a - all

+ for add
- for taking away

Controlling Ownership

- `chgrp`
  - change group ownership
  - usage `chgrp groupname file`
- `chown` **you need to own the file in the first place**
  - chown owner of file
  - typically a restricted command for security and sys admin reasons
- note: typically on lab machines, students are restricted in ability to change file ownership

File Types

- basic types: ordinary files, directories
- other types
- how to determine the type of an ordinary file?
  - especially since extensions are not required
  - `file` command
  - makes use of “magic pattern” information, typically at the beginning of the file
    - `man magic`
  - example

Pattern-Matching in Filenames

- *
  - match any number of any characters
- ?
  - match any one character
- [ ]
  - match one of a set of characters
- others
- N.B.: supported by shell, and not by the file system nor application program

ls temp* (will match anything that starts with temp)
ls temp?.pdf (it will match temp and then any char.pdf)
ls *pdf (lists all pdf files)
ls temp[15].pdf (matches 1 char that comes from set [15], so any temp1… temp5.pdf)
ls .* (lists all files that start with ‘.’)
ls -d .* (will stop at the directory, will not list contents. Will list all directories that start with ‘.’)

# all directories have ‘.’ (list to itself) and ‘..’ (list to its parent)

ls t???.txt (all names that start with a ’t’ then 3 char.txt)

More Commands Related to Files

- `more`, `less`
- `diff`, `cmp` **comparing content of files (only on ASCII text), are these 2 files the same?**
- `wc` **counts things. wants to work on txt files**
- `sort` **sorts txt files**
- `uniq` **whether or not all the lines in the txt file are unique. Assumes its sorted**
- `head`, `tail` **gives begining or end of file**
- `du` **how much disk usage. -s means sum up all, -m means megabytes**
- `df` **how much free space. What files systems are mounted.**
- example

More Commands Related to Files

- changing files between operating systems
  - DOS to UNIX
    - `unix2dos` and `dos2unix` on `tuxworld`
    - `tr '\r' '\n'`
    - `mtools` on `tuxworld`

Special Files

- `/dev/null` **anything you write here, the OS will get rid of it**
  - infinite sink
  - infinite source of end-of-file
  - uses
    - discarding output
    - terminating input

Working With Files

- Recall: “Everything in UNIX is a file”
  - useful hyperbole
- Every open file assigned a number called a *(file) descriptor*
  - small integer, starting at 0
  - think of it as a “pointer to an open file”
  - unique set per process

Standard Files

- 3 files automatically associated with every process, with every command invoked by the shell
- *stdin* - the input stream
- *stdout* - the output stream
- *stderr* - the error (output) stream
- in C, we will see these as `stdin`, `stdout`, and `stderr` defined by the `stdio` library
- in C++, we have objects `cin`, `cout`, `cerr` described as part of `cstdio`

stdin

- file descriptor 0
- default binding to the keyboard
- by default, inputs to a program are read from *stdin*

stdout

- file descriptor 1
- default binding to the terminal or display
- by default, program output is written to *stdout*

stderr

- file descriptor 2
- default binding to the terminal or display
- by default, any error or warning messages are (supposed to be) written to *stderr*
  - some programs do not comply; watch out!
  - in our code, error and warning messages should be sent to *stderr*, not *stdout*

Redirection Revisited

- earlier, saw examples like
  `cat < source > destination`
- in `bash` can explicitly redirect file descriptors by preceding the '>' or '<' with the file descriptor number
  - e.g. above equivalent to
    `cat 0< source 1> destination`
- saw `2> file` for redirection of *stderr* earlier

Redirection Revisited

- can also duplicate a file descriptor
  - two file descriptors will refer to the same file
  - full semantics beyond the scope of this course
- in `bash`, accomplished by adding ‘&’ to redirection operators
  - e.g. `>&`*`n`* instead of >

Redirection Revisited

- common use: to redirect both stdout and stderr to a single file
  - e.g. in `bash`

    ```
    prog >log 2>&1
    ```

- evaluation is left-to-right
- note: order of evaluation is important!

  ```
  prog 2>&1 >log
  ```

  does something different.

What is a Process?

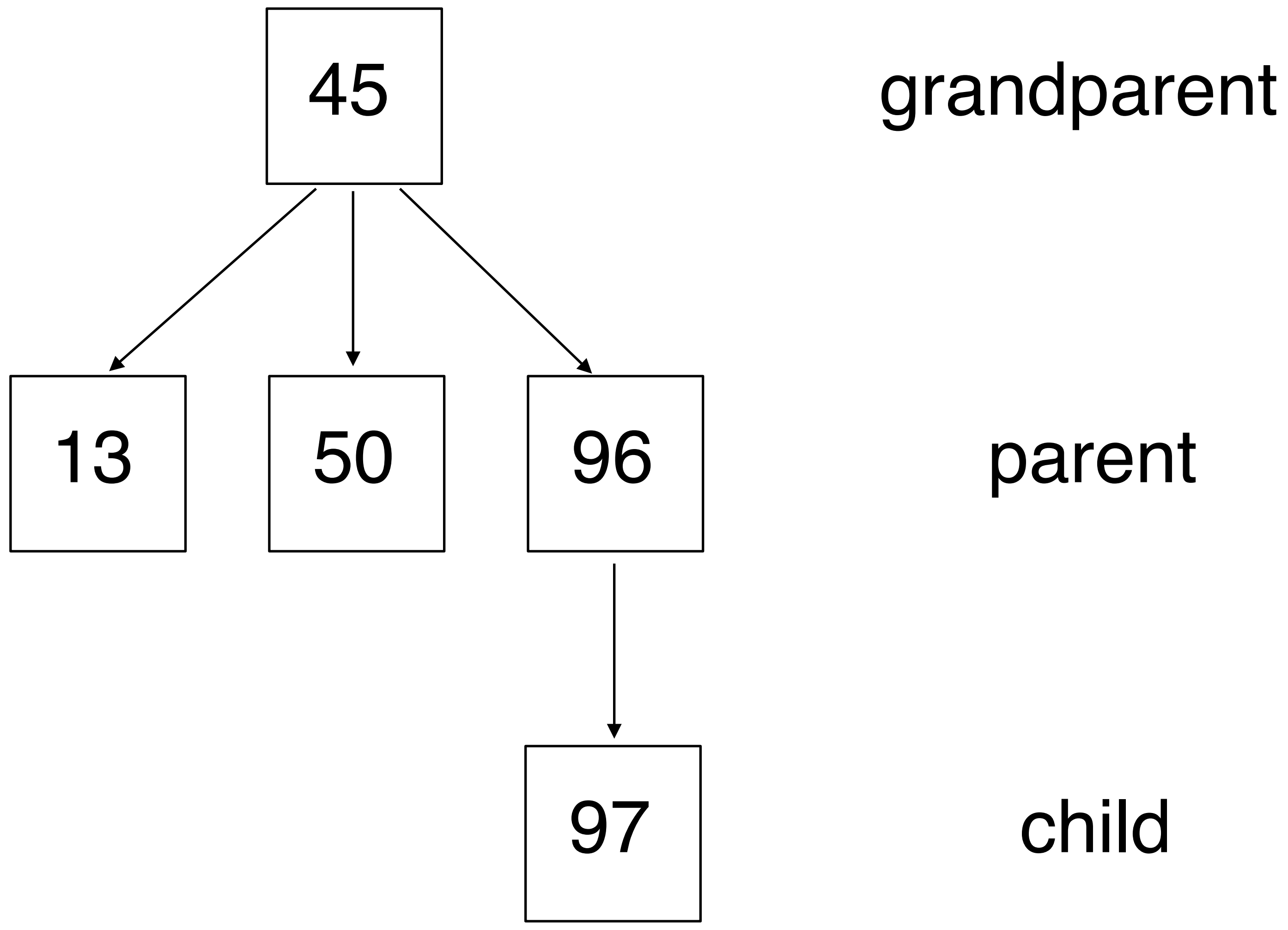
- one definition of a *process*
  - a thread of control in an address space
- recall:
  - a program my invoke several processes
  - a single process can run multiple programs

Basic Process Abstraction in UNIX

- processes exist in a hierarchy
- parent/child/sibling model
  - each process has a unique parent
  - processes can have multiple children
    - each child will be a sibling of the other children
- each process identified by a unique identifier, its *PID*

Basic Process Abstraction in UNIX

- example

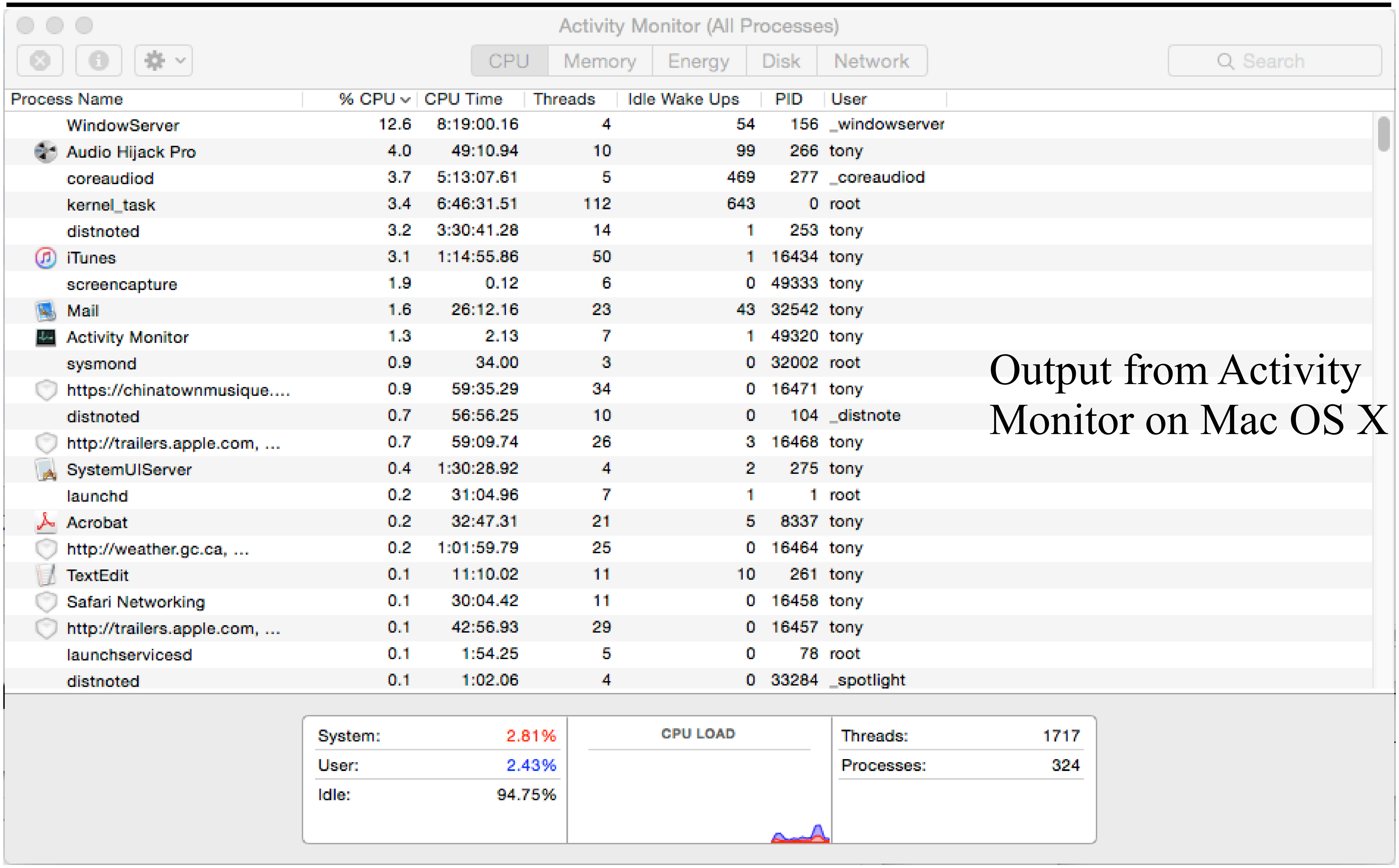


Basic Process Abstraction in UNIX

- abstraction in other operating systems is similar

```
COMMAND         PID        USER    TIME  %KER  %USE  PRI     RSS    SWAP   %MEM  THRD   %CPU
top            2620 administra    0:00   100     0    8    2076     676   0.20     1  40.00
lsass           672 SYSTEM        1h42    28    71    9   80240   77764   7.66    56   0.20
mstsc          2128 administra   25:12    48    51    8    5928    8504   0.57    10   0.12
cmd            1528 administra    0:00    71    28    8    1512    1424   0.14     1   0.05
services        660 SYSTEM        6:56    46    53    9  136580    4372  13.03    20   0.01
dns            1976 SYSTEM        5:24    53    46    8    7428    9064   0.71    14   0.01
mmc            2712 administra    0:08    62    37    8   16464    9108   1.57     5   0.01
svchost        1340 SYSTEM        4:47    41    58    8   24116   17340   2.30    41   0.01
winlogon       2884 SYSTEM        0:05    16    83   13    6412    6028   0.61    15   0.01
winlogon        600 SYSTEM        3:27    57    42   13    4796    7116   0.46    22   0.01
perl           1644 administra    2:16    24    75    8   15720    9752   1.50     4   0.00
dfssvc         1944 SYSTEM        1:52    49    50    8    4724    1892   0.45    11   0.00
svchost        1180 -             1:32    64    35    8    3652    1340   0.35    10   0.00
explorer       3540 administra    1:26    79    20    8   18172    8588   1.73     8   0.00
spoolsv        1720 SYSTEM        1:10    34    65    8    7796    5196   0.74    17   0.00
csrss          1520 SYSTEM        0:01    65    34   13    3024    1076   0.29    11   0.00
explorer        424 administra    0:00    69    30    8   10800    6368   1.03    10   0.00
```

Basic Process Abstraction in UNIX



Activity Monitor (All Processes)

Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	PID	User
WindowServer	12.6	8:19:00.16	4	54	156	_windowserver
Audio Hijack Pro	4.0	49:10.94	10	99	266	tony
coreaudiod	3.7	5:13:07.61	5	469	277	_coreaudiod
kernel_task	3.4	6:46:31.51	112	643	0	root
distnoted	3.2	3:30:41.28	14	1	253	tony
iTunes	3.1	1:14:55.86	50	1	16434	tony
screencapture	1.9	0.12	6	0	49333	tony
Mail	1.6	26:12.16	23	43	32542	tony
Activity Monitor	1.3	2.13	7	1	49320	tony
sysmond	0.9	34.00	3	0	32002	root
https://chinatownmusique....	0.9	59:35.29	34	0	16471	tony
distnoted	0.7	56:56.25	10	0	104	_distnote
http://trailers.apple.com, ...	0.7	59:09.74	26	3	16468	tony
SystemUIServer	0.4	1:30:28.92	4	2	275	tony
launchd	0.2	31:04.96	7	1	1	root
Acrobat	0.2	32:47.31	21	5	8337	tony
http://weather.gc.ca, ...	0.2	1:01:59.79	25	0	16464	tony
TextEdit	0.1	11:10.02	11	10	261	tony
Safari Networking	0.1	30:04.42	11	0	16458	tony
http://trailers.apple.com, ...	0.1	42:56.93	29	0	16457	tony
launchservicesd	0.1	1:54.25	5	0	78	root
distnoted	0.1	1:02.06	4	0	33284	_spotlight

System:	2.81%
User:	2.43%
Idle:	94.75%

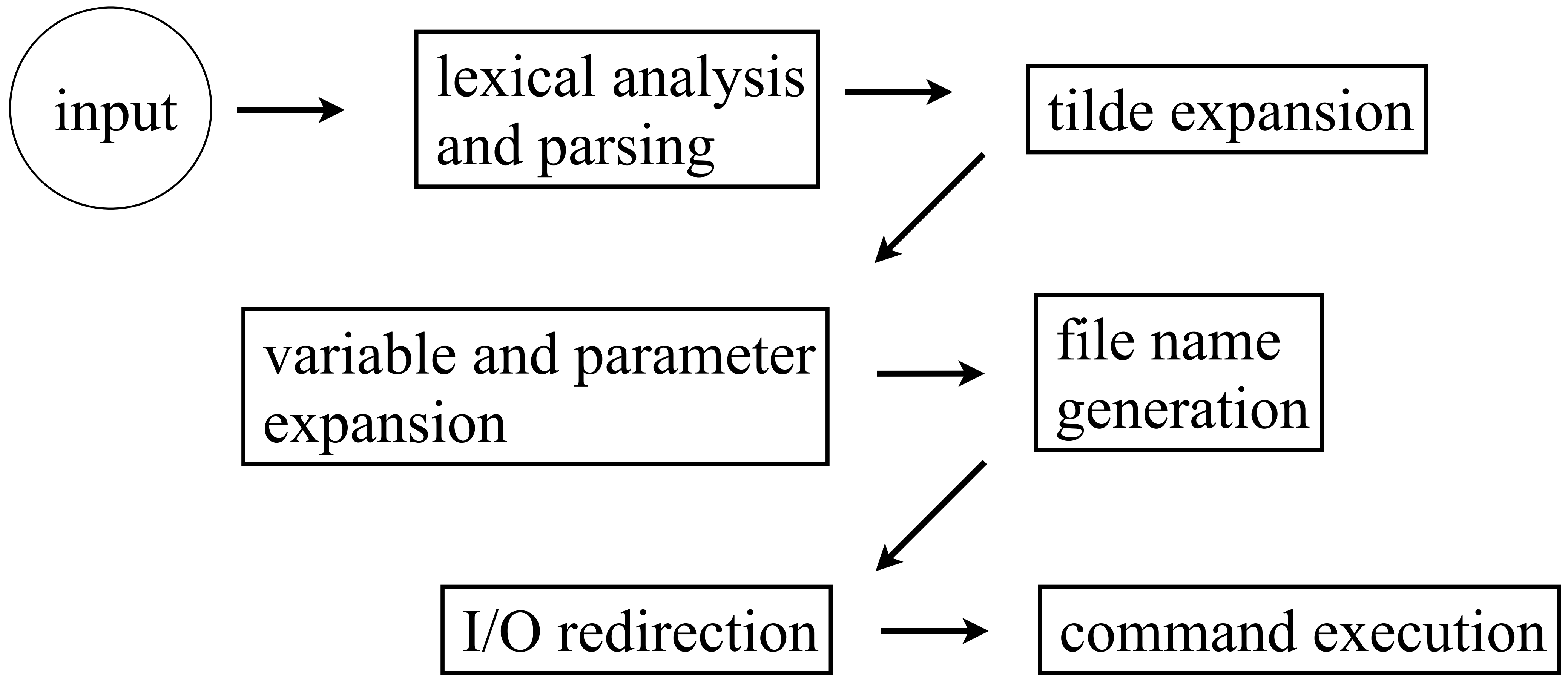
CPU LOAD

Threads:	1717
Processes:	324

Output from Activity Monitor on Mac OS X

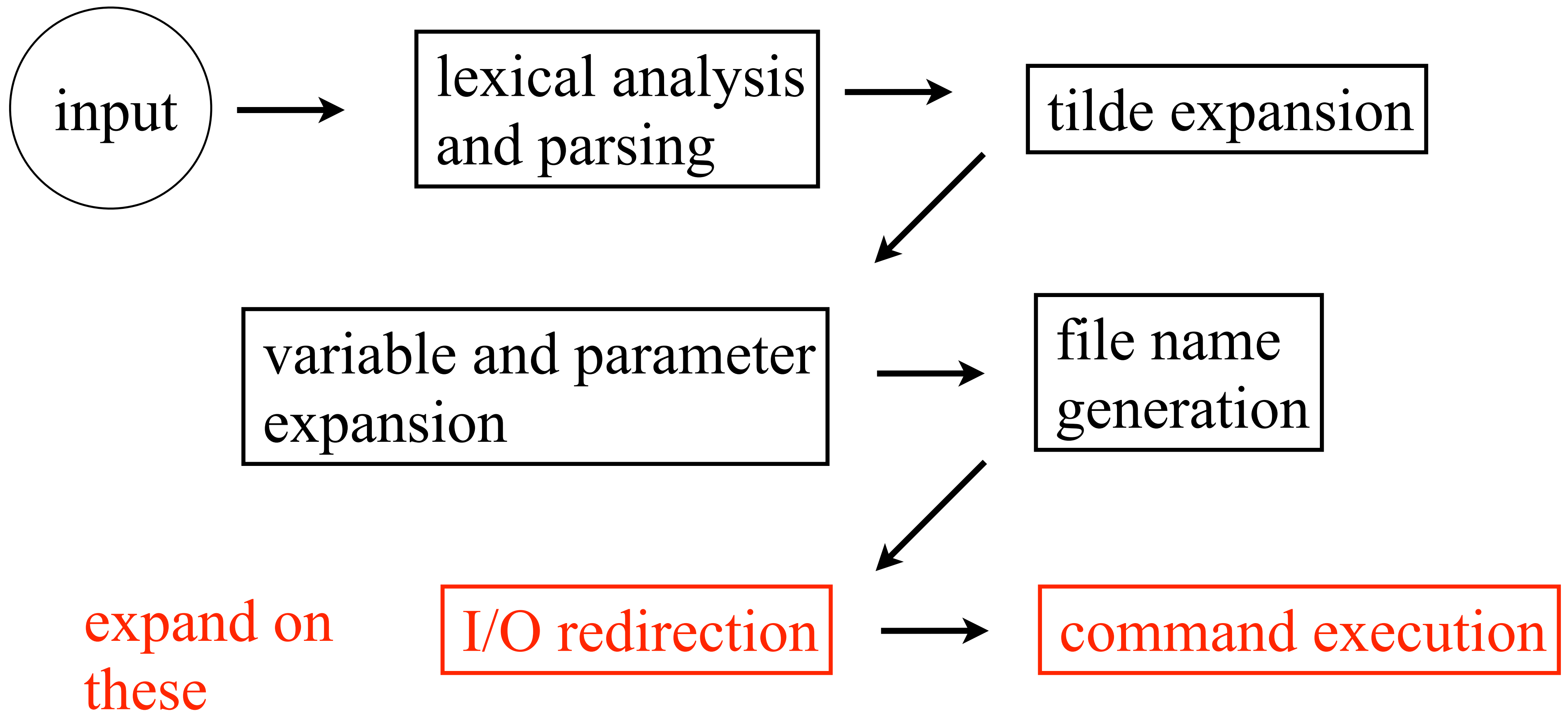
Basic Process Abstraction in UNIX

- process abstraction involved in executing a command from the shell
  - for simplicity many stages not shown



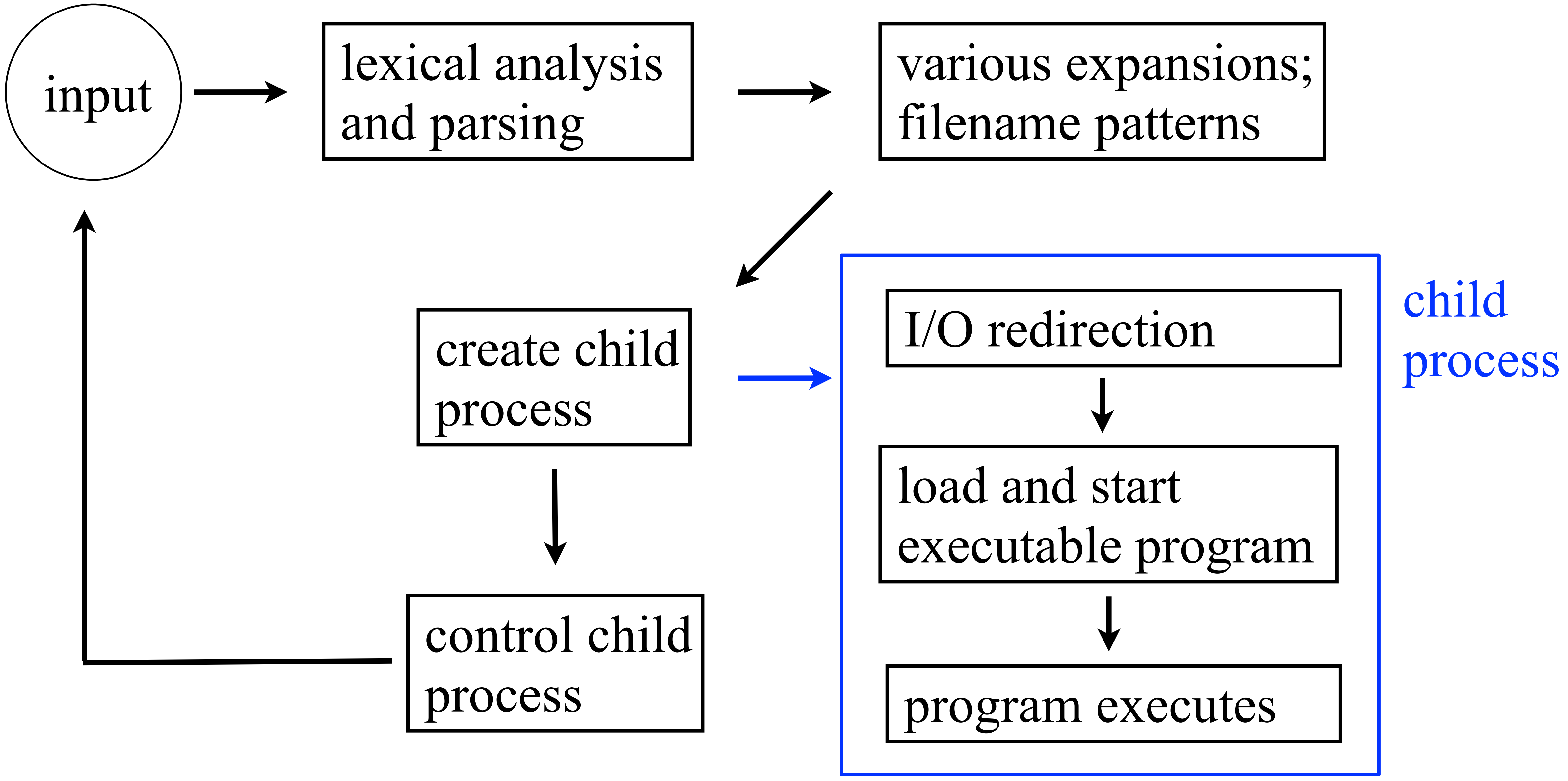
Basic Process Abstraction in UNIX

- process abstraction involved in executing a command from the shell
  - for simplicity many stages not shown



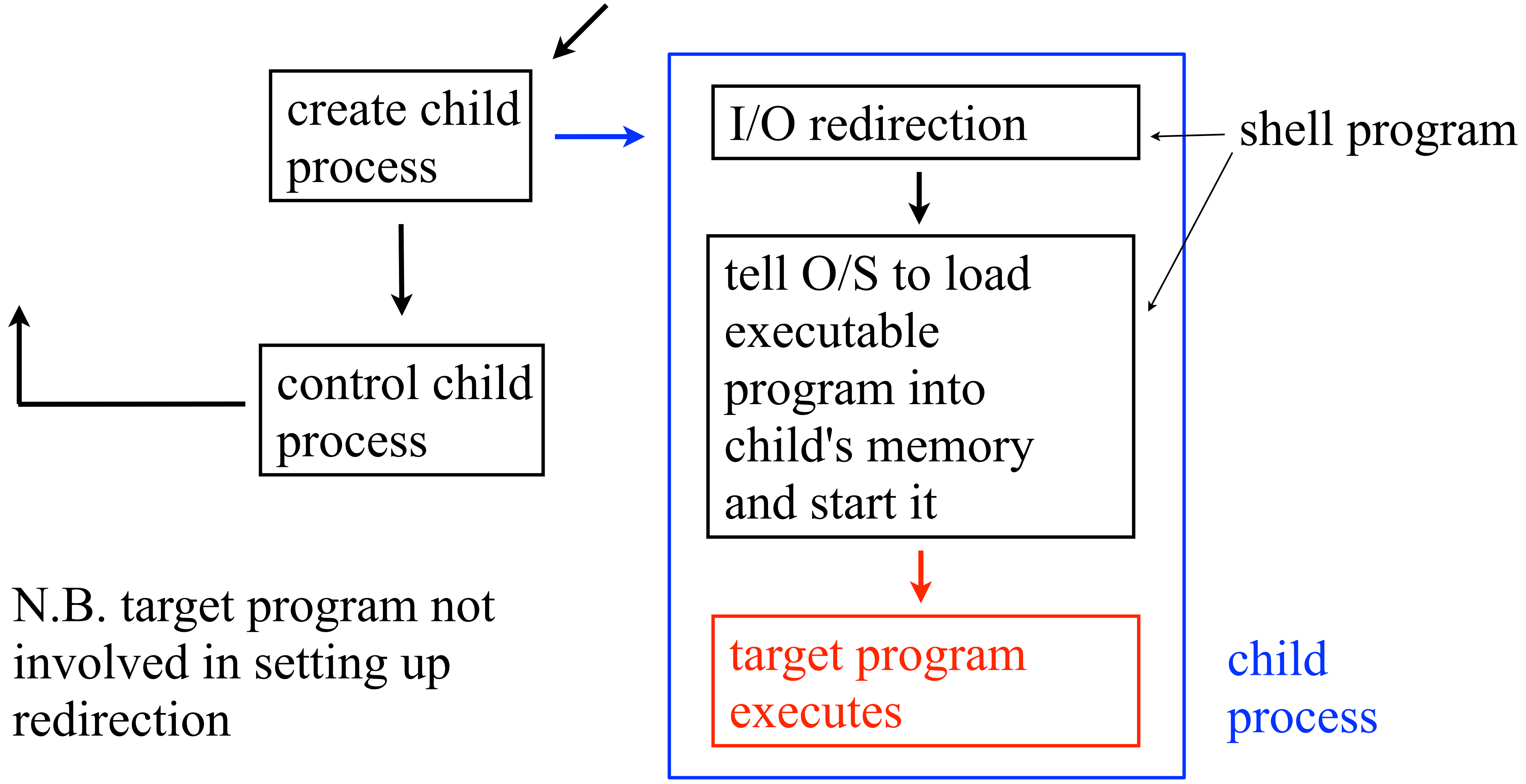
Basic Process Abstraction in UNIX

- process abstraction involved in executing a command from the shell


input
lexical analysis and parsing
various expansions; filename patterns
create child process
control child process
I/O redirection
load and start executable program
program executes
child process

Basic Process Abstraction in UNIX

- process abstraction involved in executing a command from the shell



N.B. target program not involved in setting up redirection

Commands Related to UNIX Processes

- list processes
  - `ps`
  - `pstree -h` **on** `tuxworld`
  - `top`
- `uptime`
- `w` **and** `who`
- `exit` (built-in) and `^D` (end-of-file)

Commands Related to UNIX Processes

- eliminate processes
  - `kill`
  - `man 7 signal`
  - signals generated by keyboard action: `SIGINT`, `SIGQUIT`
  - useful signals for users: `SIGKILL`, `SIGTERM`
  - `kill` built-in for `csh`, `/usr/bin/kill` or `/bin/kill` for `bash`
  - `man 1 kill` or `info kill`

Processes and Jobs

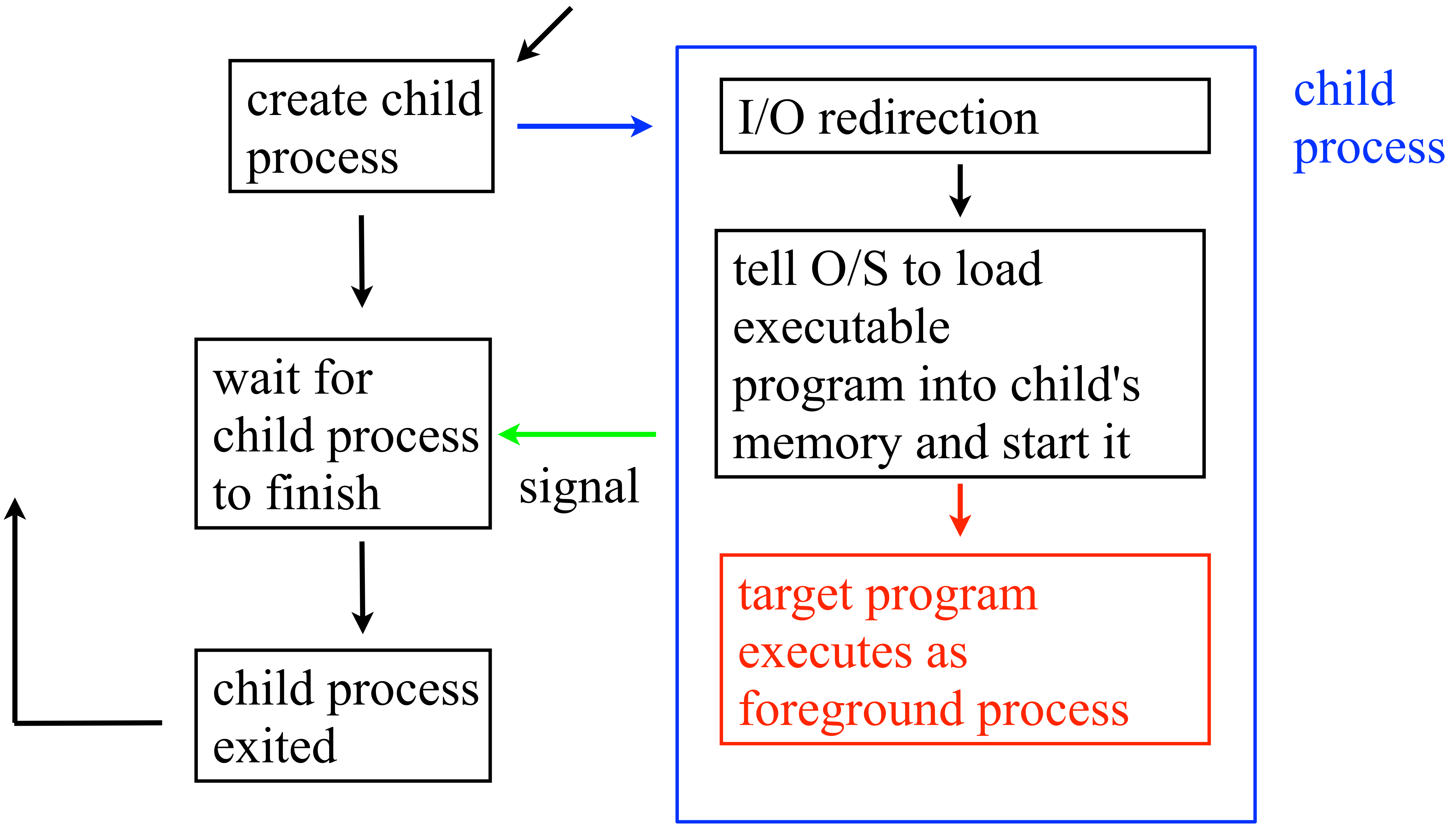
- warning: UNIX shell specific definitions
- *foreground* process:
  - a process that is associated with user input
  - usually means “has control of the keyboard”
  - shell waits for its completion
- *background* process:
  - a process that executes whenever permitted by the OS
  - usually means “does not require user interaction”
  - shell does not wait for its completion

Processes and Jobs

- suspended process:
  - a process that was executing, as permitted by the OS, but is now inactive
  - usually means “was consuming computing resources, but is no longer doing so”. However, the process is still likely using memory resources (e.g. RAM)
  - `ps -l` (LINUX) or `ps -av` (BSD)
- *job*:
  - a suspended or background running process
  - `jobs`

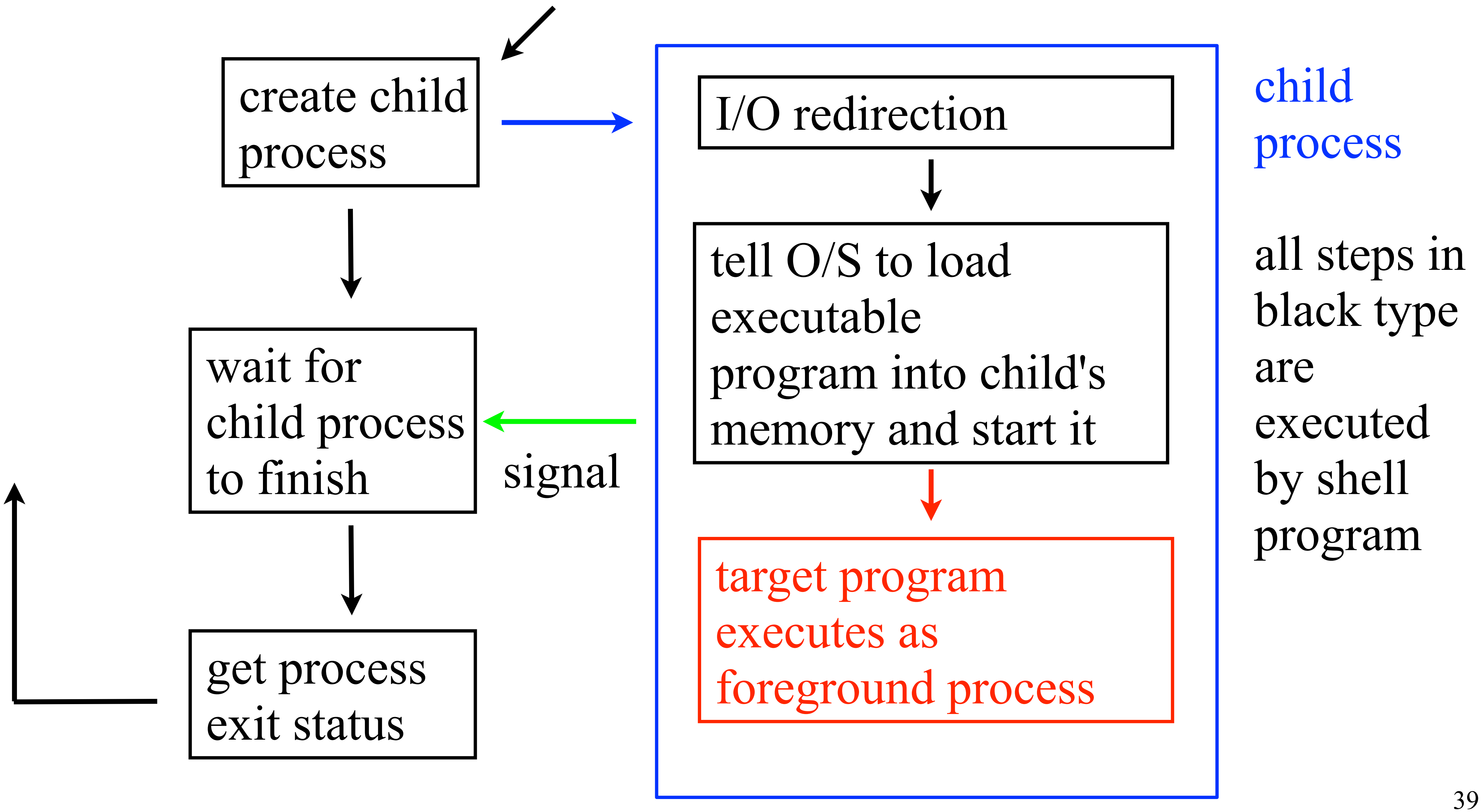
Basic Process Abstraction in UNIX

- process abstraction involved in executing a foreground command from the shell


create child process
wait for child process to finish
child process exited
signal
I/O redirection
tell O/S to load executable program into child's memory and start it
target program executes as foreground process
child process

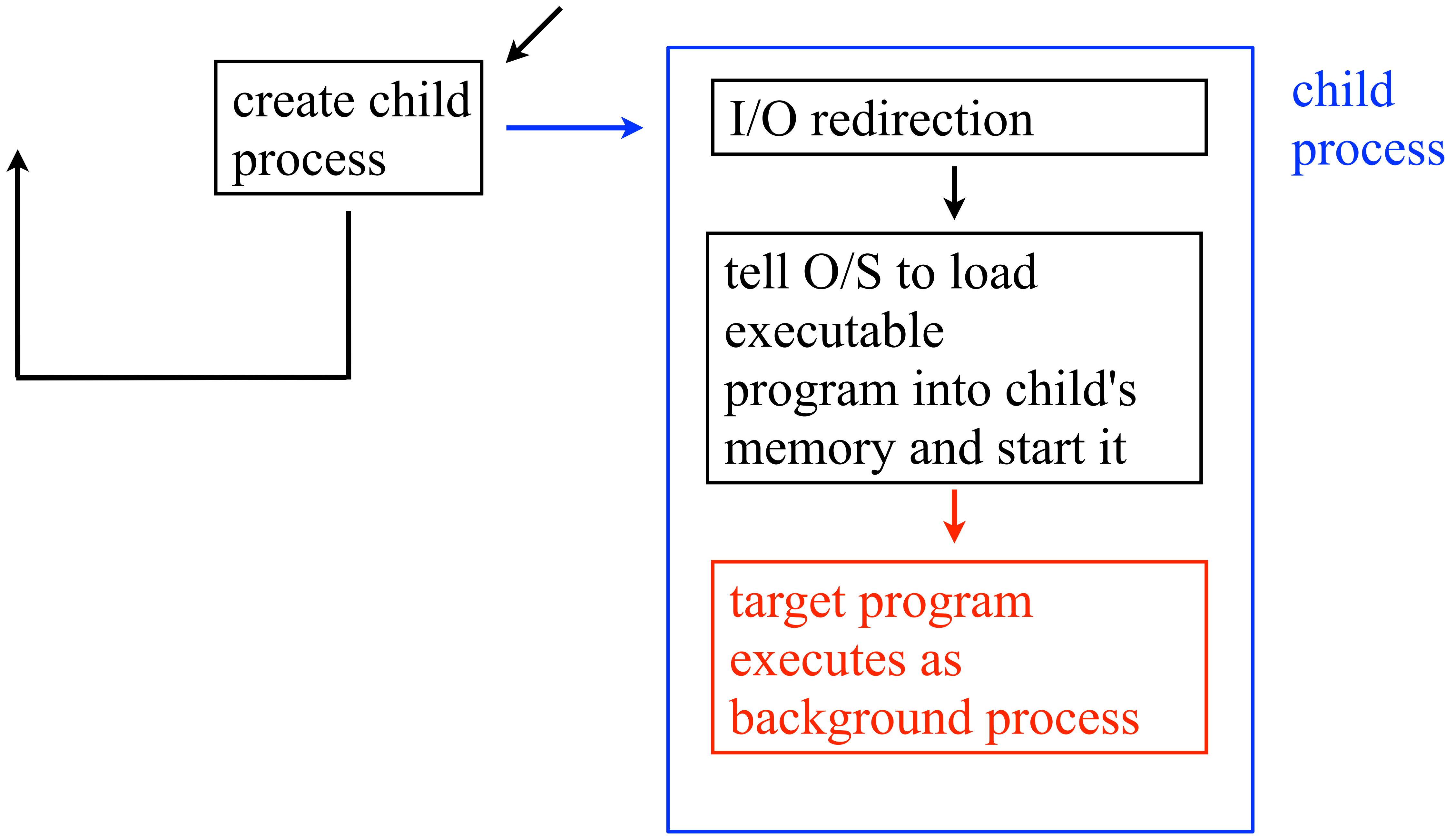
Basic Process Abstraction in UNIX

- process abstraction involved in executing a foreground command from the shell


create child process
I/O redirection
tell O/S to load executable program into child's memory and start it
target program executes as foreground process
wait for child process to finish
signal
get process exit status
child process
all steps in black type are executed by shell program

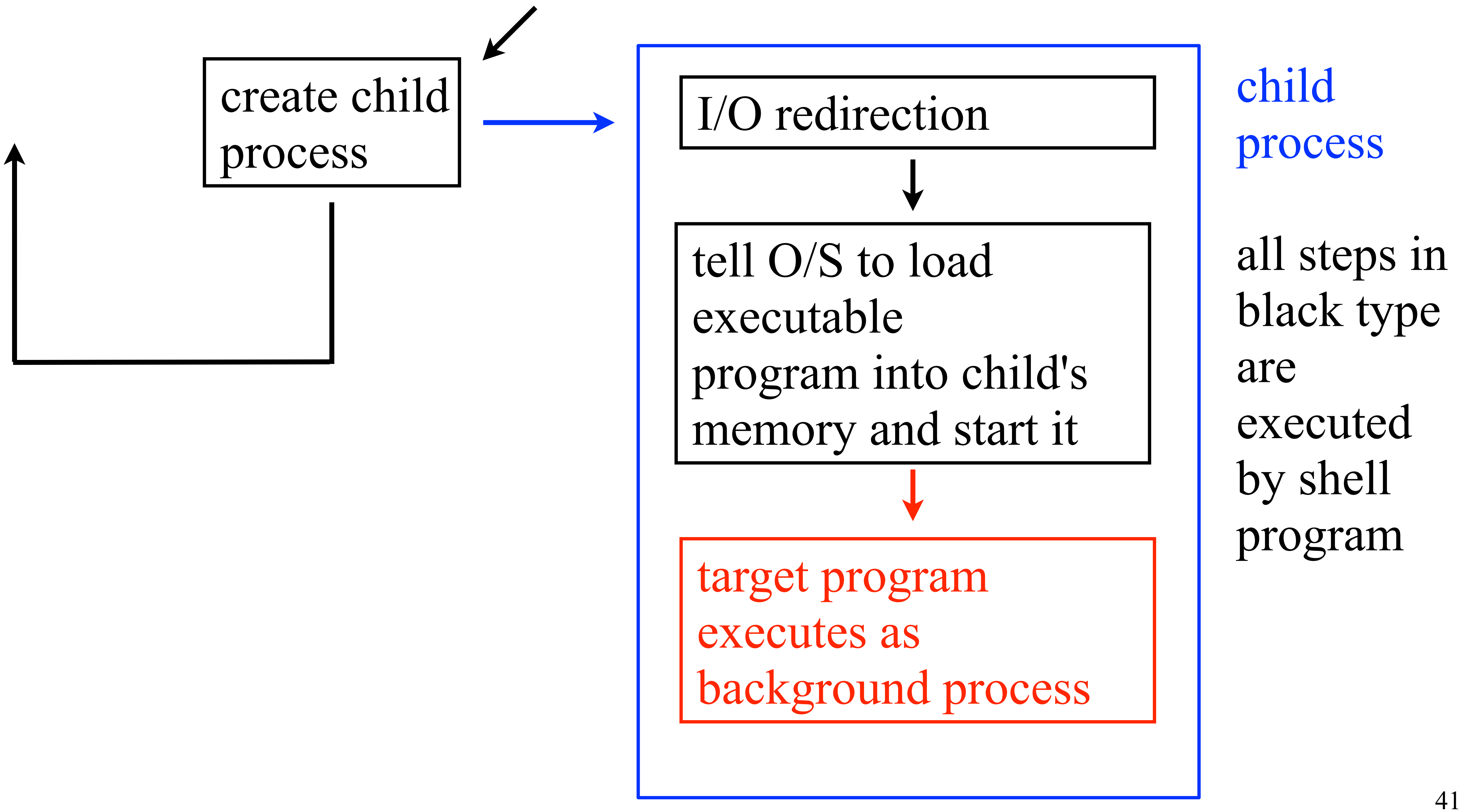
Basic Process Abstraction in UNIX

- process abstraction involved in executing a background command from the shell


create child process
I/O redirection
tell O/S to load executable program into child's memory and start it
target program executes as background process
child process

Basic Process Abstraction in UNIX

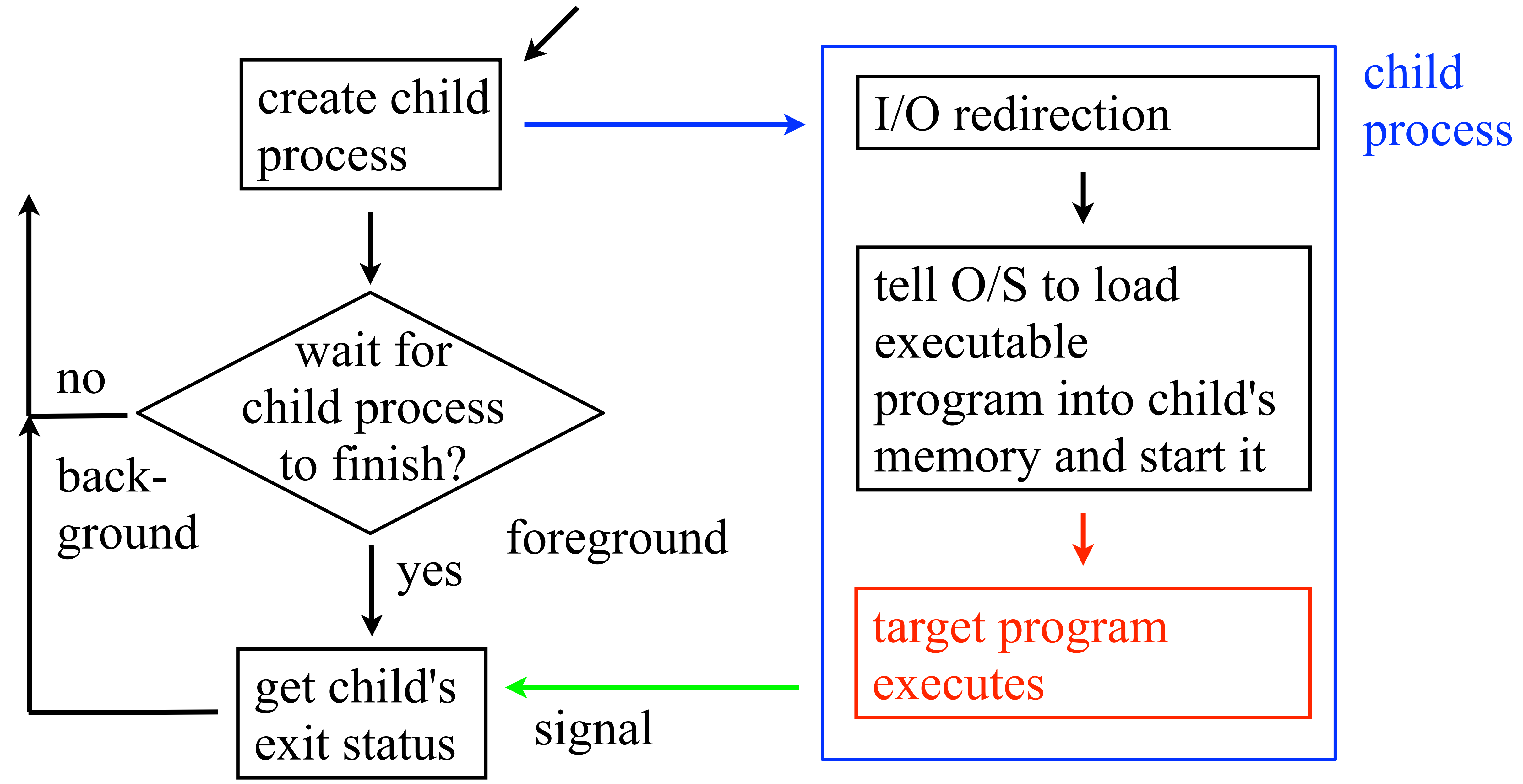
- process abstraction involved in executing a background command from the shell



create child process
I/O redirection
tell O/S to load executable program into child's memory and start it
target program executes as background process
child process
all steps in black type are executed by shell program

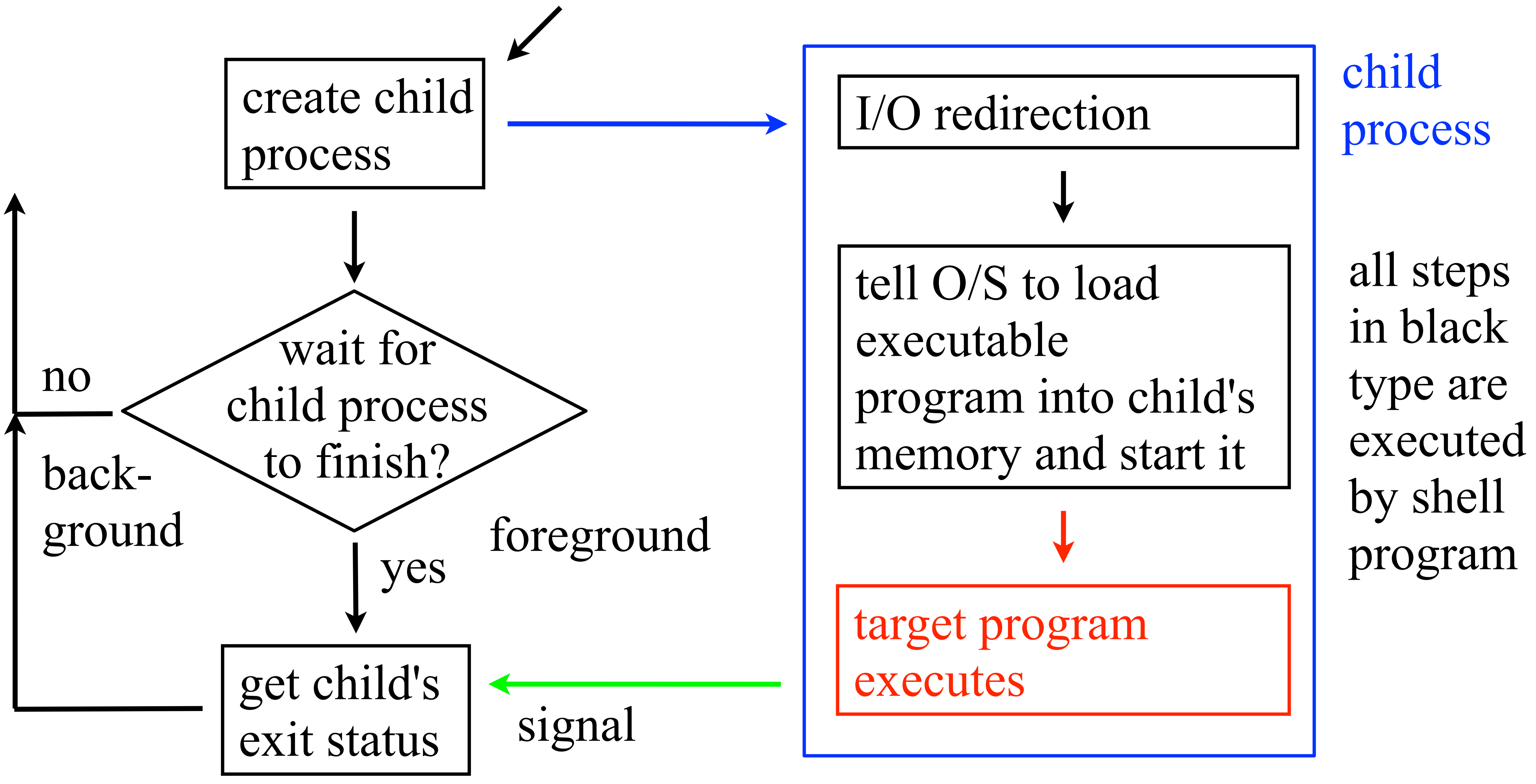
Basic Process Abstraction in UNIX

- process abstraction involved in executing a command from the shell



Basic Process Abstraction in UNIX

- process abstraction involved in executing a command from the shell


create child process
wait for child process to finish?
no
back-ground
foreground
yes
get child's exit status
signal
I/O redirection
tell O/S to load executable program into child's memory and start it
target program executes
child process
all steps in black type are executed by shell program

Processes and Jobs

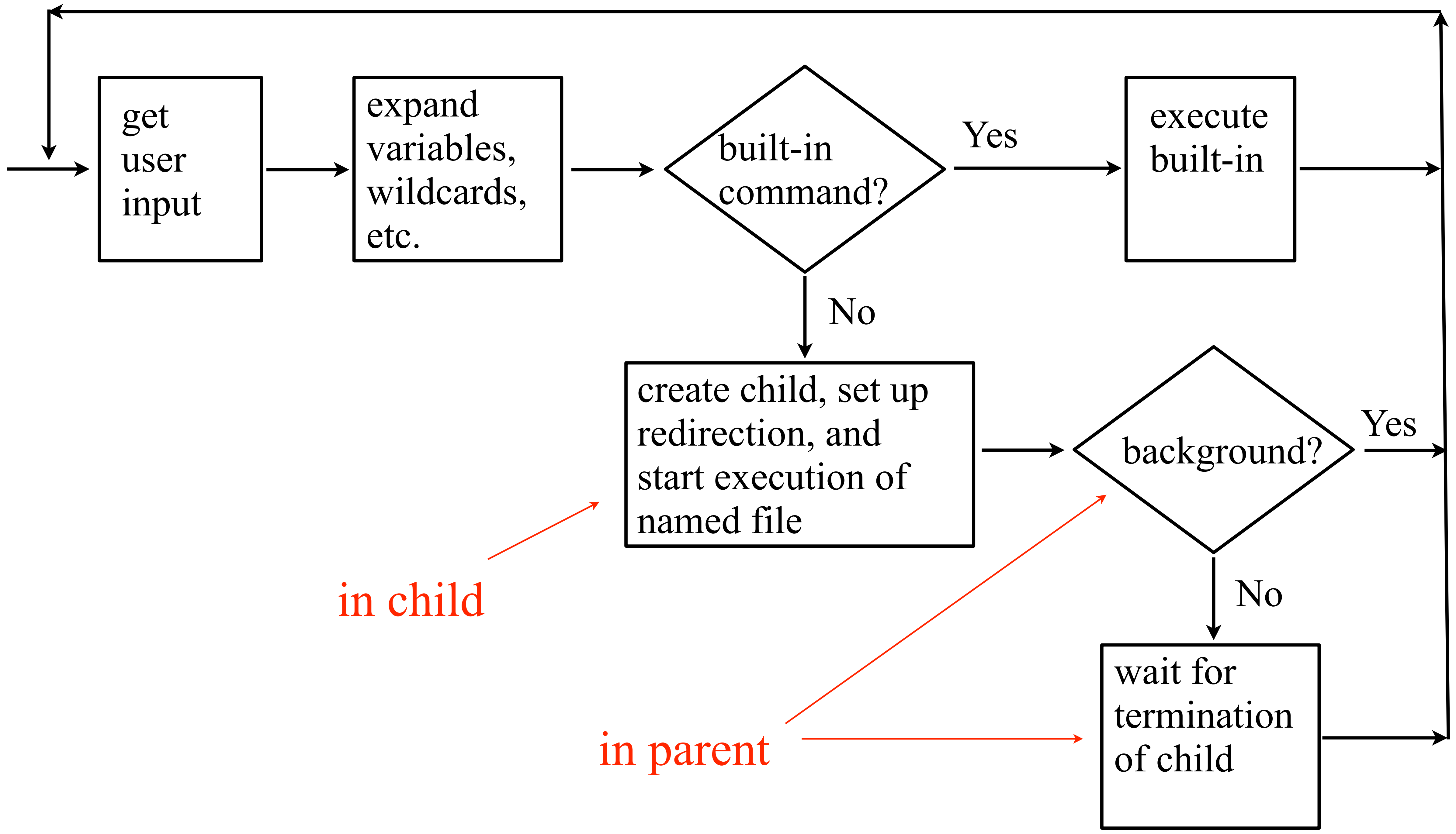
- background process
  - `&`
  - `bg`
- suspend process:
  - `^Z`
- job:
  - `jobs`
  - `%`*n*
- foreground process:
  - `fg %`*n*

Processes and Jobs

- example involving `&`, `kill`, `uniq`

```
# /bin/bash
yes > raw_stuff &
kill -TERM %1
uniq -c raw_stuff
wc -l raw_stuff
rm raw_stuff
```

Basic Shell Operation


get user input
expand variables, wildcards, etc.
built-in command?
Yes
execute built-in
No
create child, set up redirection, and start execution of named file
background?
Yes
No
wait for termination of child
in child
in parent

leave LINUX/UNIX shell … for now