SWE2024

SUNGKYUNKWAN UNIVERSITY

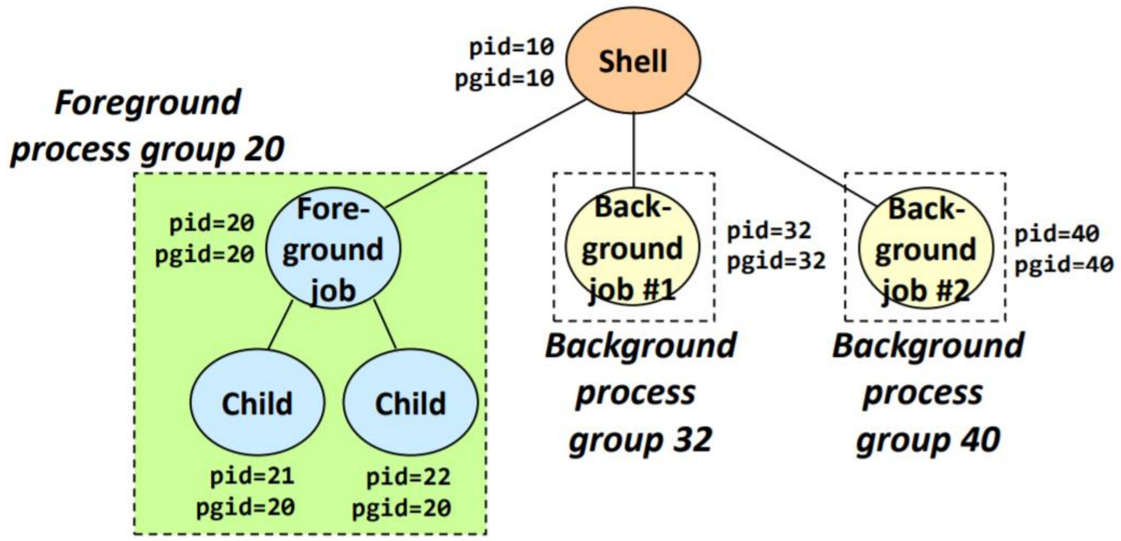
Daemon Programming

Prof. Joonwon Lee (joonwon@skku.edu)
TA – Jaehyung Park (jaeseanpark@gmail.com)
TA – Luke Albano (lukealbano@arcs.skku.edu)
Sungkyunkwan University

Process Execution Type

There are two types of processes:

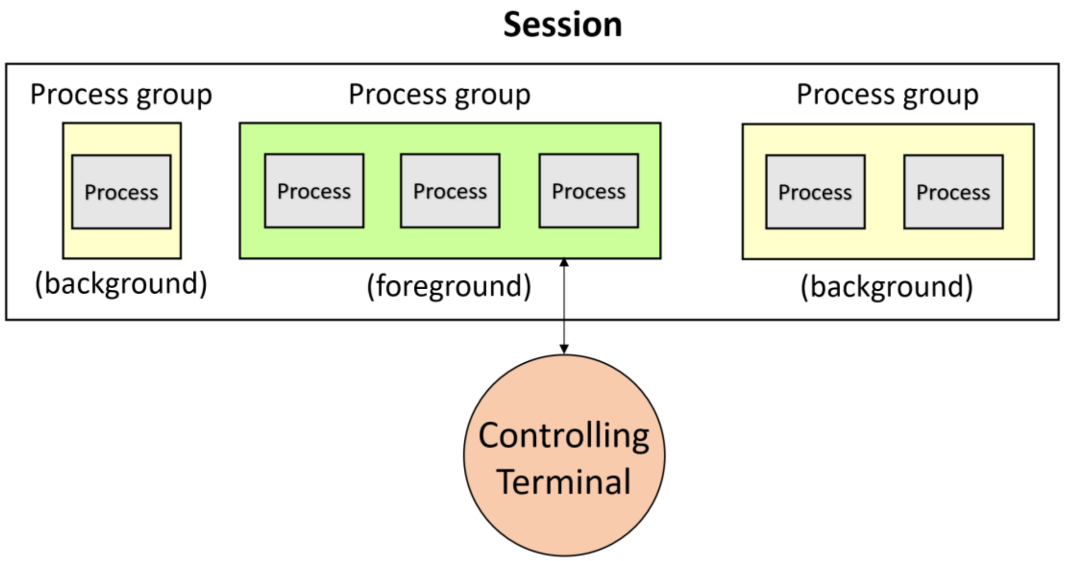
- Foreground process
  - Shell must wait for process termination
- Background process
  - Shell does not wait for process termination
  - One way to generate a background process is by appending the '&' symbol to the end of the command line



Session

Collection of one or more process groups

- Sessions can have a single terminal
  - This terminal is called a controlling terminal
- Process group within a session can be divided into:
  - A single foreground process group
  - One or more background groups



What is a Daemon?

Daemon: special type of background process

- Has no controlling terminal
  - This is because a process with a controlling terminal can be terminated unintentionally (e.g. logout or ctrl+c)
- Normally starts at system boot and keeps running forever
  - Some daemons can be launched from the user terminal
- Traditionally ends with letter ‘d’
  - e.g. sshd, crond
- Called services in Windows

Examples of Daemons

Daemon Name	Function
syslogd	Logging system facility
sshd	Incoming SSH connection service
ftpd	Incoming FTP connection service
crond	Running jobs on a pre-determined schedule
atd	Scheduling jobs with *at* command
inetd or xinetd	Managing Internet-based services
httpd	Handling HTTP requests

Daemon Coding Rules (1)

Generate a daemon process

- Call **`fork()`** and **`exit()`** system call from parent process
  - Parent process is just a role to create a daemon process
  - This is a prerequisite for creating a new session for a daemon

Daemon Coding Rules (2)

Create a new session

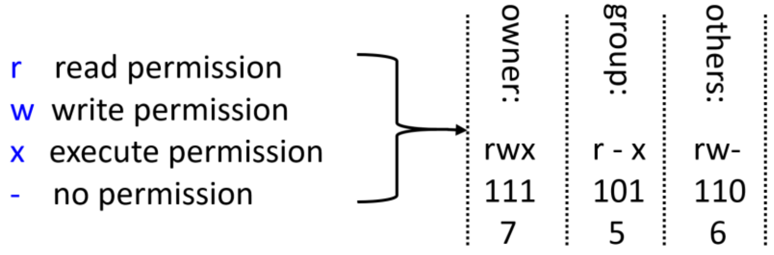
- **`pid_t setsid(void)`**
  - Makes process the leader of the process group and session
  - Returns the session ID of the calling process when it **runs successfully**
  - Returns -1 if when an **error occurs**

```
if(fork() == 0) {
   printf("old session id : %d\n", getsid(getpid()));
   if(setsid() == -1) printf("setsid failed");
   else printf("new session id : %d\n", getsid(getpid()));
}
wait(NULL);
```

Daemon Coding Rules (3)

Setting a file mode mask (**umask**)

- If the daemon process creates files, it may want to set specific file permissions
- If umask value is 1, clear value; otherwise, keep value
  - e.g. value 5 (101), umask 3 (011) ➔ 4 (100)



file permission (declared) :	756
umask (denied permission) :	037
actual file permission :	740

Bitwise AND with
The negated umask

Daemon Coding Rules (4)

- Change the current working directory to the root directory
  - Current working directory can be unmounted
- Unneeded file descriptor should be closed
  - Prevent the daemon from holding any open descriptors that it may have inherited from its parent
- Change standard file descriptors 0, 1, 2 to ***/dev/null***
  - Daemon would not require **STDIN**, **STDOUT**, **STDERR**
  - Many library functions assume that the first three descriptors are open

Daemon Coding Example

```
void main() {
    unsigned int pid;
    int fd0, fd1, fd2;

    if((pid = fork()) != 0) exit(0);
    if(setsid() < 0) exit(0);
    if(chdir(“/”) < 0) exit(0);

    umask(0);

    close(0); close(1); close(2);

    fd0 = open(“/dev/null”, O_RDWR);
    fd1 = open(“/dev/null”, O_RDWR);
    fd2 = open(“/dev/null”, O_RDWR);

    while(1) { /* contents */ }
    return 0;
}
```

Logging for Daemon Errors

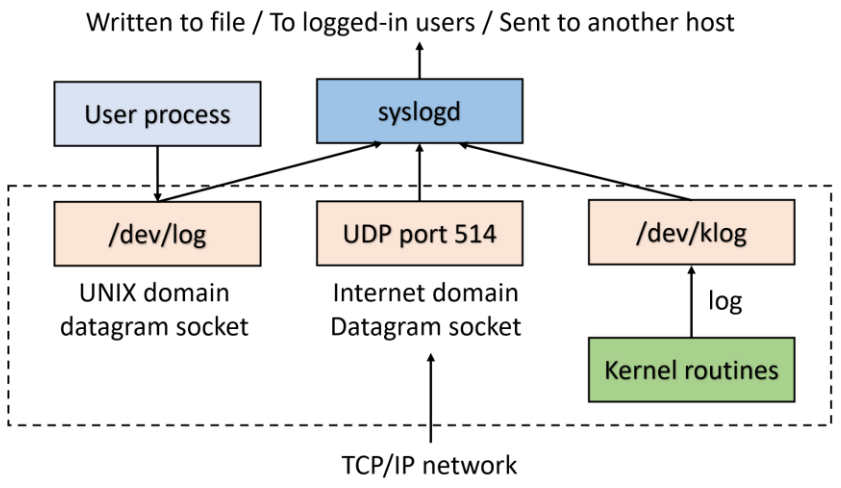
- **How to handle error messages in daemon?**
  - Daemons don't have a controlling terminal
    - It can't simply write to standard error
  - A central daemon error-logging facility is required!

- **Solution**
  - ***syslog*** daemon

syslog Daemon

There are three ways to communicate with syslogd:

1. Call the **`syslog()`** function
2. Send log messages to UDP port 514
   - Only applies to user processes that are connected to the host by a TCP/IP network
3. Kernel routines


Written to file / To logged-in users / Sent to another host
User process
syslogd
/dev/log
UNIX domain datagram socket
UDP port 514
Internet domain Datagram socket
/dev/klog
log
Kernel routines
TCP/IP network

Open & Close System Logger

```c
#include <syslog.h>

void openlog(const char *ident, int option, int facility);
void closelog(void);
```

- **`openlog()` opens a connection to the system logger**
  - `ident` is the name of the program
  - `option` is a bitmask specifying various options
  - `facility` is used to specify what type of program is logging the message
    - Refer to the configuration file (/etc/rsyslog.d)
    - Logs with different facilities are written to different files

- **`closelog()` closes the file descriptor being used to write to the system logger.**

Options for **`openlog()`**

Option	Function
LOG_CONS	Write directly to the system console if there is an error while sending to the system logger
LOG_NDELAY	Open the connection immediately
LOG_NOWAIT	Don't wait for child *processes* that may have been created while logging the message
LOG_ODELAY	Opening of the connection is delay until *syslog*() is called
LOG_PERROR	Additionally log the message to stderr
LOG_PID	Include the caller's PID with each message

Syslog Function

```
#include <syslog.h>

void syslog(int priority, const char *format, ...);
int setlogmask(int maskpri);
/* Returns : previous log priority mask value */
```

- **`syslog()` generates a log message**
  - `priority` is a combination of the facility and a level
  - `format` argument and any remaining arguments are passes to **`vsprintf()`** for formatting
- **`setlogmask()` sets the log priority mask for the process**
  - Returns the previous mask value

Syslog Facility & Priority

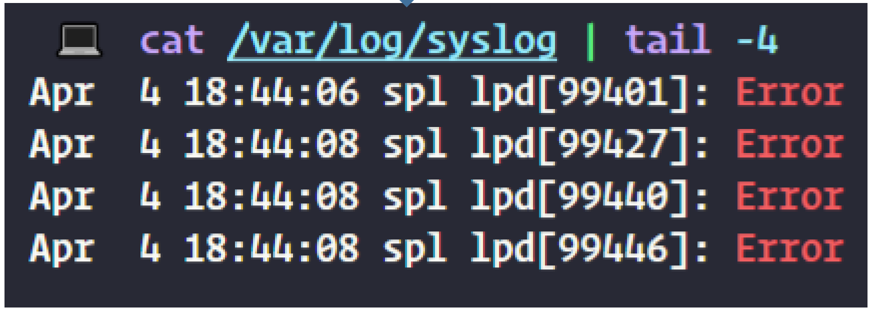
Facility	Program type
LOG_AUTH	Authorization
LOG_CRON	Cron daemon
LOG_KERN	Kernel
LOG_LRP	Line printer
LOG_MAIL	Mail system
LOG_NEWS	Network news system
LOG_USER	User process
LOG_FTP	File transfer protocol

Priority	Value
LOG_EMERG	0
LOG_ALERT	1
LOG_CRIT	2
LOG_ERR	3
LOG_WARNING	4
LOG_NOTICE	5
LOG_INFO	6
LOG_DEBUG	7

Error Logging Example

```c
#include <syslog.h>

void main(void) {
    setlogmask(LOG_MASK(LOG_EMERG));
    openlog("lpd", LOG_PID, LOG_LPR);
    syslog(LOG_EMERG, "Error");
    syslog(LOG_INFO, "Logging");
    closelog();
}
```



```
 cat /var/log/syslog | tail -4
Apr  4 18:44:06 spl lpd[99401]: Error
Apr  4 18:44:08 spl lpd[99427]: Error
Apr  4 18:44:08 spl lpd[99440]: Error
Apr  4 18:44:08 spl lpd[99446]: Error
```

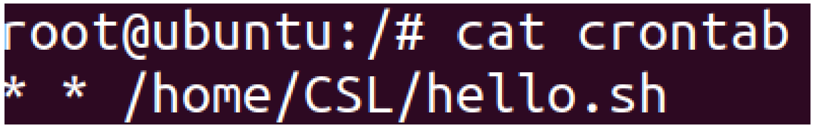
Exercise

- **Simple *cron* daemon**
  - Standard tool for running commands on a predetermined schedule
  - Automatically start when the system boots
  - Cron configuration file (*crontab*)
    - List of commands and their invocation times
    - *cron* invokes commands at predefined times

- **Make a simple *cron* daemon**

Exercise

- **Configuration file**
  - Same path as *cron* daemon execution
  - Format
    - minute (0~59), hour (0~23), executable file
  - Three arguments are separated by whitespace
    - Example:

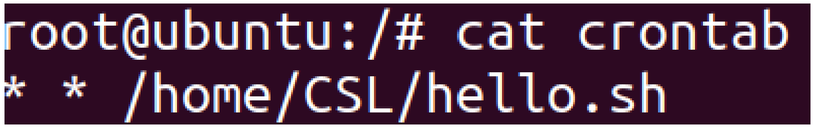


```
root@ubuntu:/# cat crontab
* * /home/CSL/hello.sh
```

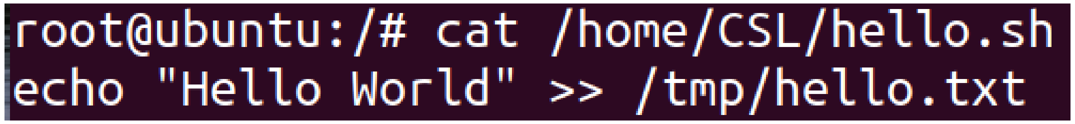
  - Rule matching
    - * matches everything
    - Any number matches exactly
    - Example:
      - * * hello.sh => executes hello.sh, every minute
      - 3 * hello.sh => executes hello.sh, 3rd minute, every hour
      - 5 4 hello.sh => executes hello.sh, 5th minute at 4am

Exercise

- **Example**
  1. Configure the ./crontab file
     - Example

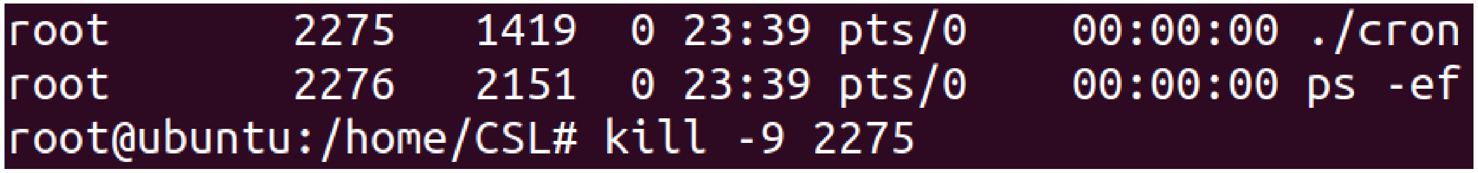


```
root@ubuntu:/# cat crontab
* * /home/CSL/hello.sh
```



```
root@ubuntu:/# cat /home/CSL/hello.sh
echo "Hello World" >> /tmp/hello.txt
```

  2. Execute simple cron daemon
  3. Terminate a cron daemon using kill command
     - kill -9 <pid>: terminate a process using process id



```
root         2275   1419  0 23:39 pts/0    00:00:00 ./cron
root         2276   2151  0 23:39 pts/0    00:00:00 ps -ef
root@ubuntu:/home/CSL# kill -9 2275
```

Exercise

- **Make simple cron daemon**
  - Get skeleton code at ~swe2024-41_23s/2023s/w6
  - You should use **struct tm *tm**
    - **tm->tm_min:** current minute
    - **tm->tm_hour**: current hour
  - The daemon should sleep until the next job is due to run
  - The daemon must reap zombie process
  - Useful API
    - **int atoi(const char* ptr):** convert a string to an integer
      - If you know how to use strtol, you can use it.
    - **unsigned int sleep(unsigned int seconds):** sleep for a specified number of seconds

Exercise hint

- **strtok_r function**

```
char str[]="System Programming Laboratory";
char *token;
char *pos = str;

while ((token = strtok_r(pos, " ", &pos)))
    printf("%s\n", token);
```

- **waitpid WNOHANG option**

- /bin/bash –c option in execl()

Exercise submission

- **Submit your source code and Makefile**
  - Via iCampus
  - Bundle source code and Makefile with tar command
    - tar.gz format
    - tar cvzf [student_id].tar.gz [all your files]
  - We will compile by using command ***make***
    - If compilation fails, your points for this exercise will be **zero**
  - Due today