Reparenting -- why to avoid (waitpid usage)

execl

./a.out 10 20 abc 2.3 xyz

```
int s1;
waitpid(pid, &s1, 0);
WIFEXITED(s1), WEXITSTATUS(s1)
```

```
execl("./a.out", "a.out", "10",
"20", "abc","2.3", "xyz",NULL);
```

```
ret=fork();
if(ret==0)
        //child code
else
        //parent code
```

```
char* argv[5] = {"a.out", "10",
"20", "abc","2.3", "xyz", NULL};

execv("./a.out", argv);
```

```
if(ret==0) {
   k=execl("/bin/date","date",NULL);
}
```

```
execlp("date", "date", "+%D", NULL);
```

exit is a system call in kernel space
_exit is wrapper for exit system call

exit is a library call in userspace

TODO:- exit vs _exit (exit - lib call, _exit - sys call wrapper)

```
for(i=1;i<=10;i++)
   printf("hello:%d\t", i);                    //no \n, skipping flush
exit(0);    //_exit(0);                        //??
```

Zombie Process / Zombie State.

exit/_exit:- release all resources except PCB and process table entry
zombie state

parent waitpid:- clean up PCB and process table entry
terminated state

process table is of finite length

```
ulimit -a
ulimit -u

ps
ps aux
ps -el
ps -e -o pid, ppid, stat, cmd

pstree
pstree -np

top                         # q - quit
```

Threads:-

Multitasking

Multithreading

* Concurrent execution
* Resource Sharing (same address space)

* increased utilization of CPU

Threads will share all resources, except stack
Every thread maintains private stack

fork vs thread create
==> independent vs shared resources
==> thread creation is faster than fork (no duplication of resources)

Threads are known as Light Weight Process (LWP)

Concurrency Examples (Real Applications)
* Office Suite
* browser
* streaming media player
* concurrent server
* parallel sum of large array

Threads -- independent scheduling attributes & context
-- common resource attributes (memory, fs, i/o)

Thread Models:-
* User level threads, managed by userspace library (thread manager)
kernel is not aware of threads
many to one mapping
e.g. Traditional UNIX

* Kernel level threads, directly managed by Kernel
one to one mapping
e.g. Linux, Modern OS

ps -eL -o pid,ppid,lwp,nlwp,stat,cmd

Every process runs as single thread initially (LWP)

POSIX Thread Library [not system calls]

pthread_create
pthread_exit
pthread_join

pthread_equals
pthread_self
---
prototyped in pthread.h

defined in pthread libs
(libpthread.a or libpthread.so)

/lib, /usr/lib etc

```
void* do_work1(void* pv) {
   printf("Thread A -- Welcome\n");
   for(i=1;i<=max;i++)
        printf("A--%d\n",i);
}
//write do_work2 similar to do_work1

pthread_t pt1;
pthread_t pt2;
pthread_create(&pt1, NULL, do_work1, NULL);
pthread_create(&pt2, NULL, do_work2, NULL);
----

gcc psimple.c -lpthread

exit(0)     ==>       release all resources
            ==>       all threads will terminate

pthread_exit    ==> only current thread will terminate
                ==> no release of resources
```

pthread_join:-
  * block till the completion of specified thread
  * collect exit status of completed thread (for whom waited)

1st param:- pthread_t variables
2nd param:- to collect exit status

pthread_create:-
1st param:- addr of pthread_t variable
2nd param:- attributes, NULL means default
3rd param:- addr/name of service function (callback)
4th param:- arguments to service functions

```
for(i=1;i<=max;i++)              //max can be 10 or 20
{
   printf("A--%d\n",i);
   sleep(1);                     //usleep
}
```

```
void* task_body(void* pv)
{
        char* ps=pv;
        -----
}

pthread_create(&pt1,NULL,task_body,"A1");
pthread_create(&pt2,NULL,task_body,"B2");
pthread_create(&pt3,NULL,task_body,"C3");
-------------------
int k;
pthread_t ptarr[n];
for(i=0;i<n;i++)
{
   k=100+i;
   pthread_create(&ptarr[i],NULL,task_body,(void*)k);
}

void* do_service(void* pv) {
   int k = (int)pv;

}
```

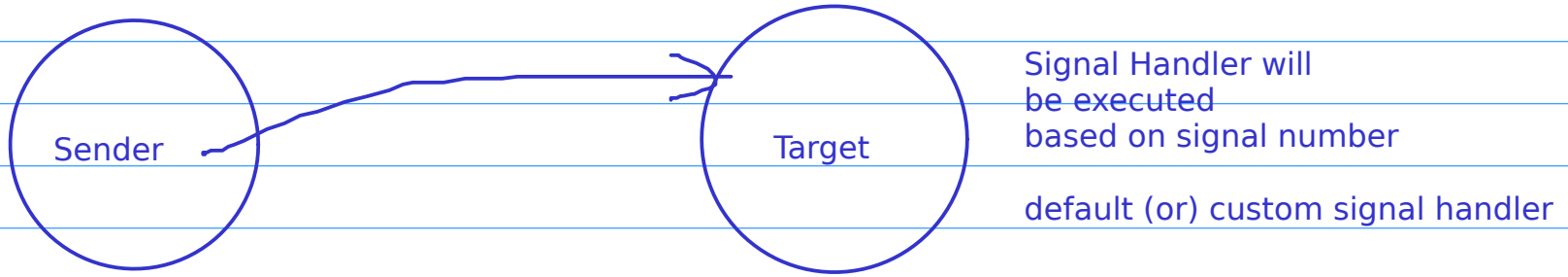
Task Driven Concurrency
Data Driven Concurrency

Signal Handling:-

Signals -

Asynchronous event .. from one process to other
(software level, kernel level)

one process will generate/trigger signal (sender)
signal will reach other process (target)



Most default handlers cause abnormal termination of a process

familair signal generations.
ctrl + C ==> interrupting process (abnormal term) ==> SIGINT
ctrl + \ ==> SIGQUIT
ctrl + z ==> SIGTSTP
kill <pid> ==> SIGTERM , soft kill
kill -9 <pid> ==> SIGKILL , sure kill
child exit ==> SIGCHLD (to parent)

some other :- SIGSTOP, SIGCONT, SIGFPE, SIGSEGV, SIGALRM

kill -l # list of signals

kill -INT <pid>
kill -2 <pid>
pkill -INT <pname>
pkill -2 <pname>
killall
---
pkill(pid, signo); # system call
pkill(0, signo); # sending signal to same process

foreground process ==> stdin, stdout focus to user
background process ==> detached from stdin
==> instead of stdout, logging techniques

./a.out # f/g
./a.out& # b/g

command& # run in b/g, no stdin, logging instead of stdout/stderr

jobs ==> list of background and suspended processes

fg ==> resume suspended or b/g process in f/g
bg ==> resume suspended process in b/g

fg <job-id>
bg <job-id>
if no job-id, recent job will be chosen

echo $? # exit status of recent command

signal API (system call) -- register custom handler
pause -- block until any signal arrives
raise -- send to signal to self
alarm -- sending signal after specified no.of seconds (SIGALRM)

In PCB ==> signal pending field (collection of bits)
==> signal mask field (collection of bits)

mask bit 0 ==> signal will be delivered and handled
mask bit 1 ==> signal can't be delivered (ignored/masked)
pending bit 0 ==> such signal not arrived
pending bit 1 ==> signal will be handled (default/custom handler)

signal(SIGxxx, handler); //replace default handler with
//custom handler

SIGKILL, SIGSTOP ==> non maskable signals
==> custom handler not applicable
==> only default handler (sure kill, sure suspend)

Modern signal APIs:-
* sigaction
* sigsuspend
* sigprocmask
* sigwait
* kill

```
void handler_for_alarm(int signo) {
  //print current time
  //Hints:- time system call, ctime lib API
  alarm(1);
}

signal(SIGALRM, handler_for_alarm);
alarm(5);
while(1)
pause();
```

Scheduling

TODO:-

* post read of Threads, Signals + hands-on, examples
* pre-read on scheduling
* pre-read on IPC,Semaphores
* Start Analyzing Assignment-1