Homework1

Name Yu Yang   Student ID 892449550

3.1

The output is still 5, the child just holds a copy of value.

3.2

8, it constructs a process tree.

3.9

The kernel must save the context of the old process and restore the saved context of the process scheduled to run. Saving the context of a process includes the values of all CPU registers, in addition to memory allocated to it(program counter, list of open files….)

3.18

a. Synchronous and asynchronous communication
Sync communication allows a rendezvous between sender and receiver. Disadvantage of blocking is rendezvous might not be need because sender has no interest to know when receives.
b. Automatic and explicit buffering
Automatic buffer provide queue with indefinite size, so the sender don't have to block and wait message is removed. But memory is waisted. Explicit one specifies how large buffer is. Sender may be blocked in explicit buffering. It is less likely memory is wasted in explicit one.
c. Send by copy and send by reference
Send by copy doesn't allow receiver modify the original parameter while by reference does. But passing by reference requires declaring the parameter as a remote parameter as well.
d. Fixed-sized and variable-sized messages
This is related to buffering problems. With fixed size, a buffer can hold specific number of message while not in variable-sized.

4.6

Sequential problems are not fit for multithread. Such as calculating an individual tax return or shell programs such as opening files.

4.7

When one kernel thread suffers from page fault, another kernel thread can be switched in to use the interleaving time in a useful manner.
And when a program has to wait for other system events, multithread would give better performance.

4.8

b. Heap memory and c. Global variables

4.24

For this simple program, I limit the number of prime to 200.

```
#include <stdio.h>
#include <zconf.h>
#include <pthread.h>
#include <stdlib.h>
int result[200];
void *runner(void *param){
    //para is the bound
    int para = atoi(param);
    //1 is not prime
    result[0]=0;
    result[1]=0;
    int ii,iii;
    for(ii=2;ii<para;ii++)
        result[ii]= 1;
    //any int == ii * iii, is not prime
    for(ii=2;ii<=para/2;ii++)
        for(iii=2;iii<=para/ii;ii++)
            result[ii*iii]=0;
    pthread_exit(0);
};

int main(int argc, char *ag[]) {
	if(argc != 2) {
		perror("arg num is not2,");
		exit(1);
	}
	if(atoi(ag[1])<2) {
		perror("the num should be >2");
		exit(1);
	}
	pthread_t pid;
	pthread_attr_t at;
	int i;
	//get attr
	pthread_attr_init(&at);
	//creat thread
	pthread_create(&pid,&at,runner,ag[1]);
	//wait thread finish
	pthread_join(pid,NULL);
	for(i=0;i<atoi(ag[1]);i++){
		if(result[i])
			printf("prime: %d",i);
	}
}
```

Problem 1

```
#include <stdio.h>
#include <fcntl.h>
#include <zconf.h>

int main() {
	int file = open("/dev/null",O_WRONLY);
	char *s = "1234567890";
	int i = 0;
	for(i = 0; i<100000000;++i)
		write(file,s,10);
}
```

Real 0m13.900s

```
User 1.633s
Sys 12.276s
```

Problem2

```
#include <stdio.h>
#include <fcntl.h>
#include <zconf.h>

int fake_write(int fd,const void *buffer,size_t sz){
	return sz;
}
int main() {
	//open file
	int file = open("/dev/null",O_WRONLY);
	char *s = "1234567890";
	int i = 0;
	//write
	for(i = 0; i<100000000;++i)
		fake_write(file,s,10);
}
```

```
Real 0m0.304s
User 0m0.301s
Sys 0m0.001s
```

Bacause it need to switch from usr mode to kernel mode, and when system call are done, it will switch back, this take much system time. And IO is slow.

Problem 3

exec

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
```

```
#include <sys/wait.h>

void *runner(void *param){
	if(fork()==0){
		printf("command line param is %s\n",param);
		//wait another
		wait(NULL);
	}else{
		execl("/bin/ls","ls",NULL);
		printf("exec() return -- should not be returned");
	}
	pthread_exit(0);
}
int main(int argc, char *ag[]) {
	if(argc != 2){
		perror("arg num is not 2");
		exit(1);
	}
	pthread_t pid;
	pthread_attr_t at;
	pthread_attr_init(&at);
	pthread_create(&pid,&at,runner,ag[1]);
	pthread_join(pid,NULL);
}
```

Exec doesn't copy threads, so the printf("exec() return -- should not be returned") will not be executed.

Fork

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <sys/wait.h>

void *runner(void *param){
	if(fork()==0){
		printf("command line param is %s\n",param);
		//wait another
		wait(NULL);
	}else{
		if(fork() == 0){
			printf("fork() in thread");
		}
```

```
                printf("fork() in thread 2nd\n");
        }
        printf("after fork\n");
        pthread_exit(0);
}
int main(int argc, char *ag[]) {
        if(argc != 2){
                perror("arg num is not 2");
                exit(1);
        }
        pthread_t pid;
        pthread_attr_t at;
        pthread_attr_init(&at);
        pthread_create(&pid,&at,runner,ag[1]);
        pthread_join(pid,NULL);
}
```

Fork copy thread, printf("fork() in thread 2nd\n"); are printed twice.

Threadsig.c

```
#include <stdio.h>
#include <zconf.h>
#include <pthread.h>
#include <stdlib.h>
#include <signal.h>
int result[200];
void sig_handle(int sig){
        printf("terminated!");
};

int main() {
        signal(SIGINT,sig_handle);
        sleep(100);
}
```

As written in the textbook:

" When a signal is generated by an event external to a running process, that process receives the signal asynchronously. Examples of such signals include terminating a process with specific keystrokes (such as <control><C>)and having a timer expire."