const int nprimes = 100;
int prime[nprimes];
int main() {
    int i;
    int current = 2;
    prime[0] = current;
    for (i=1; i<nprimes; i++) {
        int j;
        NewCandidate:
        current++;
        for (j=0; prime[j]*prime[j] <= current; j++) {
            if (current % prime[j] == 0)
                goto NewCandidate;
        }
        prime[i] = current;
    }
    return(0);
}
The Unix Address Space

- stack
- dynamic
- bss
- data
- text
Modified Program

```c
const int nprimes = 100;
int *prime;
int main(int argc, char *argv[]) {
    int i;
    int current = 2;
    nprimes = atoi(argv[1]);
    prime = (int *)malloc(nprimes*sizeof(int))
    prime[0] = current;
    for (i=1; i<nprimes; i++) {
        ...
    }
    return(0);
}
```
System Calls

• Sole interface between user and kernel
• Implemented as library routines
• Errors indicated by returns of −1; error code is in errno

```c
if (write(fd, buffer, bufsize) == −1) {
    // error!
    printf("error %d\n", errno);
    // see perror
}
```
Creating a Process: Before

fork( )

parent process
Creating a Process: After

parent process

fork( )  // returns p

child process (pid = p)

fork( )  // returns 0
Fork and Wait

```c
short pid;
if ((pid = fork()) == 0) {
    /* some code is here for the child to execute */
    exit(n);
} else {
    int ReturnCode;
    while (pid != wait(&ReturnCode))
        ;
    /* the child has terminated with ReturnCode as its return code */
}
```
**Exec**

```c
int pid;
if ((pid = fork()) == 0) {
   /* we’ll soon discuss what might take place before exec is called */
   execl("/home/twd/bin/primes", "primes", "300", 0);
   exit(1);
}

/* parent continues here */

while(pid != wait(0)) /* ignore the return code */
    ;
```
Loading a New Image

Before

exec(prog, args)

After

args

prog’s bss

prog’s data

prog’s text
Standard File Descriptors

```c
main( ) {
    char buf[BUFSIZE];
    int n;
    const char* note = "Write failed\n";

    while ((n = read(0, buf, sizeof(buf))) > 0)
        if (write(1, buf, n) != n) {
            (void)write(2, note, strlen(note));
            exit(EXIT_FAILURE);
        }
    return(EXIT_SUCCESS);
}
```
Back to Primes ...

```c
int nprimes;
int *prime;
int main(int argc, char *argv[]) {
    ... 
    for (i=1; i<nprimes; i++) {
        ...
    }
    if (write(1, prime, nprimes*sizeof(int)) == -1) {
        perror("primes output");
        exit(1);
    }
    return(0);
}
```
int nprimes;
int *prime;
int main(int argc, char *argv[]) {
    ...
    for (i=1; i<nprimes; i++) {
        ...
    }
    for (i=0; i<nprimes; i++) {
        printf("%d\n", prime[i]);
    }
    return(0);
}
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int options [, mode_t mode])

- options
  - O_RDONLY open for reading only
  - O_WRONLY open for writing only
  - O_RDWR open for reading and writing
  - O_APPEND set the file offset to end of file prior to each write
  - O_CREAT if the file does not exist, then create it, setting its mode to mode adjusted by umask
  - O_EXCL if O_EXCL and O_CREAT are set, then open fails if the file exists
  - O_TRUNC delete any previous contents of the file
  - O_NONBLOCK don’t wait if I/O can’t be done immediately
Running It

```c
if (fork() == 0) {
    /* set up file descriptor 1 in the child process */
    close(1);
    if (open("/home/twd/Output", O_WRONLY) == -1) {
        perror("/home/twd/Output");
        exit(1);
    }
    execl("/home/twd/bin/primes", "primes", "300", 0);
    exit(1);
}
execl("/home/twd/bin/primes", "primes", "300", 0);
exit(1);

/* parent continues here */

while (pid != wait(0)) /* ignore the return code */
    ;
```