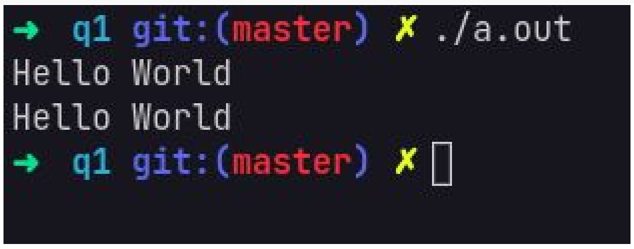
SS LAB 1

SAHIL BONDRE U18CO021

fork()

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char const *argv[]) {
  fork();
  printf("Hello World\n");
  return 0;
}
```



```
➜  q1 git:(master) ✗ ./a.out
Hello World
Hello World
➜  q1 git:(master) ✗ 
```

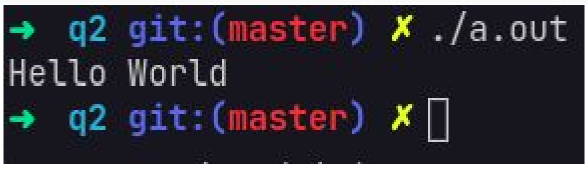
"Hello World" printed twice

exec()
hello.c

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
  printf("Hello World\n");
  return 0;
}
```

main.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char const *argv[]) {
  char *args[] = {"./hello", NULL};
  execvp(args[0], args);
  printf("End\n");
  return 0;
}
```

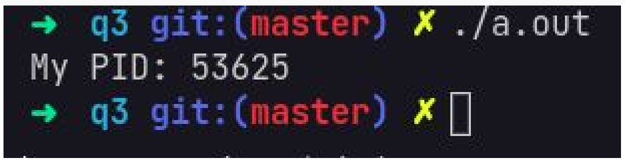


```
➜  q2 git:(master) ✗ ./a.out
Hello World
➜  q2 git:(master) ✗
```

“Hello World” printed but not “End”

getpid()

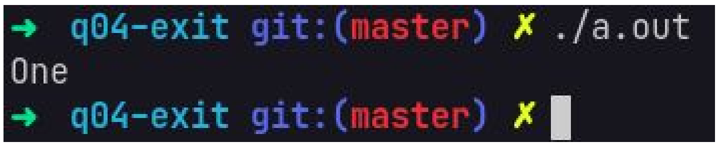
```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char const *argv[]) {
  pid_t ID = getpid();
  printf("My PID: %d\n", ID);
  return 0;
}
```



```
➜  q3 git:(master) ✗ ./a.out
My PID: 53625
➜  q3 git:(master) ✗
```

exit()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char const *argv[]) {
  printf("One\n");
  _exit(0);
  printf("Two\n");
  return 0;
}
```

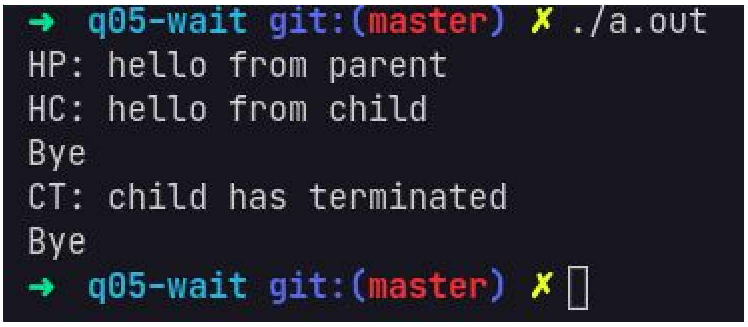


```
➜  q04-exit git:(master) ✗ ./a.out
One
➜  q04-exit git:(master) ✗
```

Only “One” printed as exit() was called before “Two”

wait()

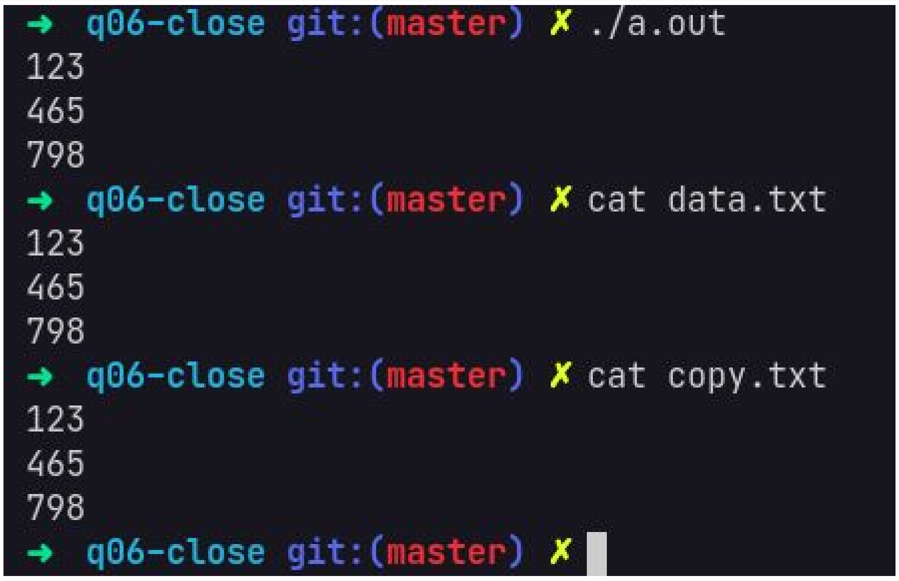
```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(int argc, char const *argv[]) {
  if (fork() == 0)
    printf("HC: hello from child\n");
  else {
    printf("HP: hello from parent\n");
    wait(NULL);
    printf("CT: child has terminated\n");
  }

  printf("Bye\n");
  return 0;
}
```



```
➜  q05-wait git:(master) ✗ ./a.out
HP: hello from parent
HC: hello from child
Bye
CT: child has terminated
Bye
➜  q05-wait git:(master) ✗ 
```

open() read() write() close() stat()

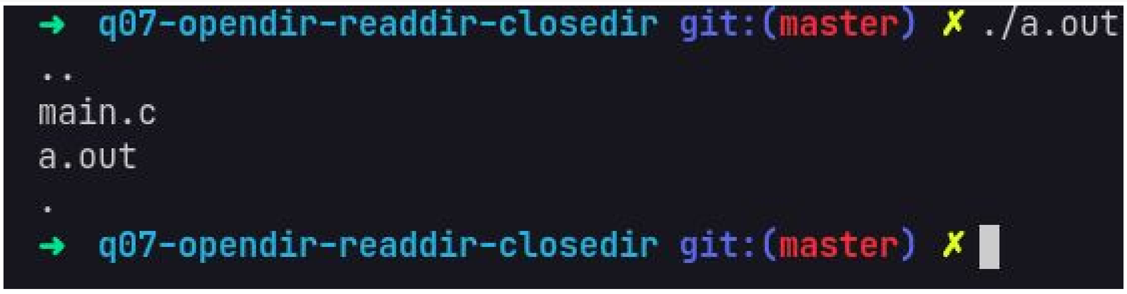
```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>

int main(int argc, char const *argv[]) {
  int fd1 = open("./data.txt", O_RDONLY);
  int fd2 = open("./copy.txt", O_WRONLY);

  if(fd1 < 0 || fd2 < 0) {
    printf("Cannot open files\n");
    exit(1);
  }
  struct stat st;
  fstat(fd1, &st);
  int size = st.st_size;

  char *c = (char *)calloc(size, sizeof(char));

  read(fd1, c, size);
  printf("%s", c);
  write(fd2, c, size);

  close(fd1);
  close(fd2);
  return 0;
}
```



```
➜  q06-close git:(master) ✗ ./a.out
123
465
798
➜  q06-close git:(master) ✗ cat data.txt
123
465
798
➜  q06-close git:(master) ✗ cat copy.txt
123
465
798
➜  q06-close git:(master) ✗
```

opendir() closedir() readdir()

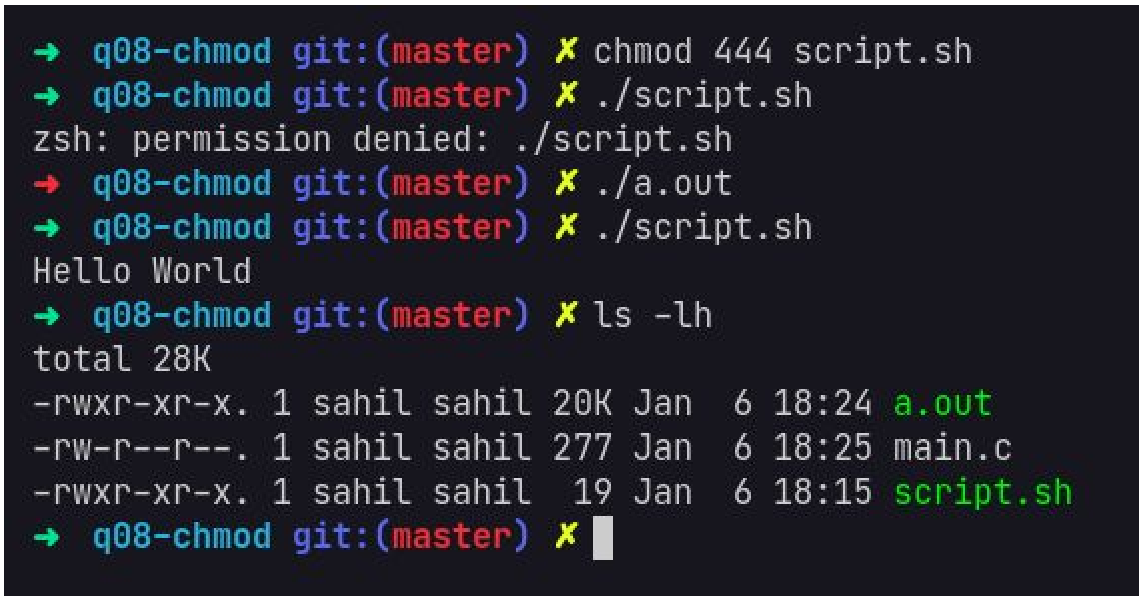
```
#include <dirent.h>
#include <stdio.h>
#include <unistd.h>

int main(void) {
  struct dirent *de;
  DIR *dr = opendir(".");

  if (dr == NULL) {
    printf("Could not open current directory");
    _exit(1);
  }

  while ((de = readdir(dr)) != NULL) {
    printf("%s\n", de->d_name);
  }

  closedir(dr);
  return 0;
}
```



```
➜ q07-opendir-readdir-closedir git:(master) ✗ ./a.out
..
main.c
a.out
.
➜ q07-opendir-readdir-closedir git:(master) ✗
```

chmod()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>

int main(int argc, char const *argv[]) {
  mode_t mode = 0755;
  uid_t owner = 01000;
  uid_t group = 01000;

  chmod("./script.sh", mode);
  chown("./script.sh", owner, group);
  return 0;
}
```

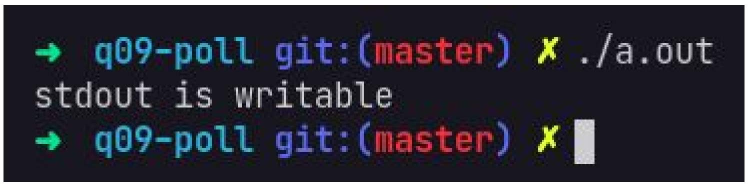


```
➜  q08-chmod git:(master) ✗ chmod 444 script.sh
➜  q08-chmod git:(master) ✗ ./script.sh
zsh: permission denied: ./script.sh
➜  q08-chmod git:(master) ✗ ./a.out
➜  q08-chmod git:(master) ✗ ./script.sh
Hello World
➜  q08-chmod git:(master) ✗ ls -lh
total 28K
-rwxr-xr-x. 1 sahil sahil 20K Jan  6 18:24 a.out
-rw-r--r--. 1 sahil sahil 277 Jan  6 18:25 main.c
-rwxr-xr-x. 1 sahil sahil  19 Jan  6 18:15 script.sh
➜  q08-chmod git:(master) ✗
```

making “script.sh” executable and setting ownership

poll()

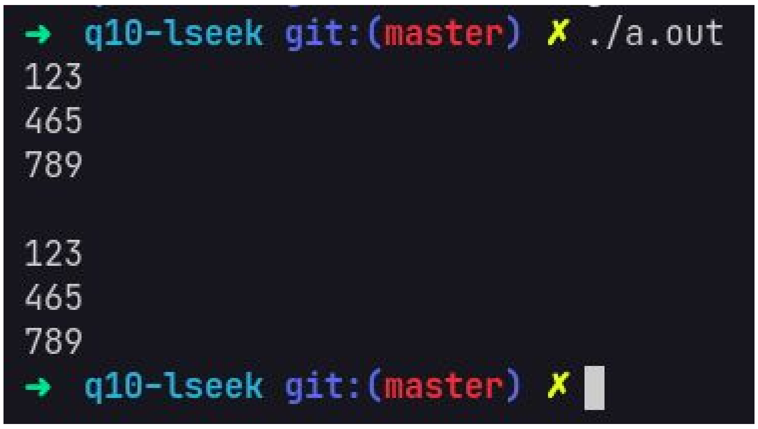
```c
#include <stdio.h>
#include <sys/poll.h>
#include <unistd.h>

#define TIMEOUT 5000

int main(void) {
  struct pollfd fds[3];
  int ret;

  /* watch stdin for input */
  fds[0].fd = STDIN_FILENO;
  fds[0].events = POLLIN;

  /* watch stdout for ability to write */
  fds[1].fd = STDOUT_FILENO;
  fds[1].events = POLLOUT;

  /* watch stderr for ability to write */
  fds[2].fd = STDERR_FILENO;
  fds[2].events = POLLERR;

  ret = poll(fds, 3, TIMEOUT * 1000);

  if (ret == -1) {
    perror("poll");
    return 1;
  }

  if (!ret) {
    printf("%d seconds elapsed.\n", TIMEOUT);
    return 0;
  }

  if (fds[0].revents & POLLIN) {
    printf("stdin is readable\n");
  }

  if (fds[1].revents & POLLOUT) {
    printf("stdout is writable\n");
  }

  if (fds[2].revents & POLLERR) {
    printf("stderr is writable\n");
  }

  return 0;
}
```



```
➜  q09-poll git:(master) ✗ ./a.out
stdout is writable
➜  q09-poll git:(master) ✗
```

lseek()

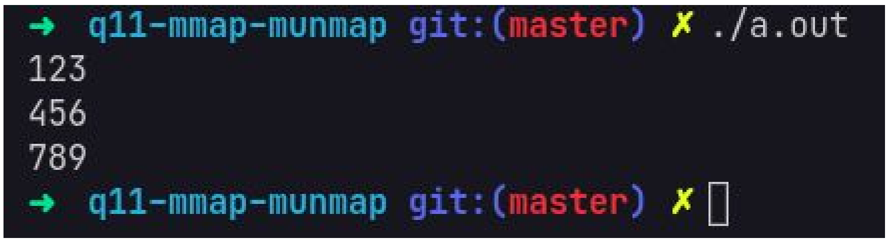
```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char const *argv[]) {
  int fd = open("data.txt", O_RDONLY);
  struct stat st;
  fstat(fd, &st);
  int size = st.st_size;
  char *c1 = (char *)calloc(size, sizeof(char));
  char *c2 = (char *)calloc(size, sizeof(char));

  read(fd, c1, size);
  printf("%s\n", c1);

  // Move back to beginning of file
  lseek(fd, 0, SEEK_SET);
  read(fd, c2, size);
  printf("%s", c2);

  close(fd);
  return 0;
}
```



```
➜  q10-lseek git:(master) ✗ ./a.out
123
465
789

123
465
789
➜  q10-lseek git:(master) ✗
```

mmap() munmap()

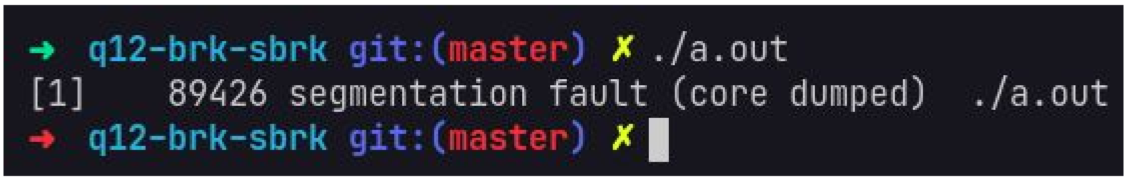
```c
#include <fcntl.h>
#include <stdio.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char const *argv[]) {
  int fd = open("./data.txt", O_RDONLY);

  struct stat st;
  fstat(fd, &st);
  int size = st.st_size;

  char *data = mmap(NULL, size, PROT_READ, MAP_PRIVATE, fd, 0);

  printf("%s", data);
  munmap(data, size);

  return 0;
}
```



```
➜  q11-mmap-munmap git:(master) ✗ ./a.out
123
456
789
➜  q11-mmap-munmap git:(master) ✗ 
```

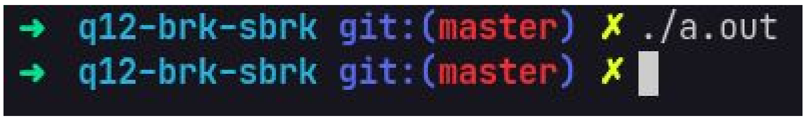
brk() sbrk()

```c
#include <unistd.h>

int main(int argc, char const *argv[]) {
  int *p = sbrk(0);

  brk(p + 4);
  *p = 1;
  return 0;
}
```



```
➜  q12-brk-sbrk git:(master) ✗ ./a.out
[1]    89426 segmentation fault (core dumped)  ./a.out
➜  q12-brk-sbrk git:(master) ✗ 
```

without brk()

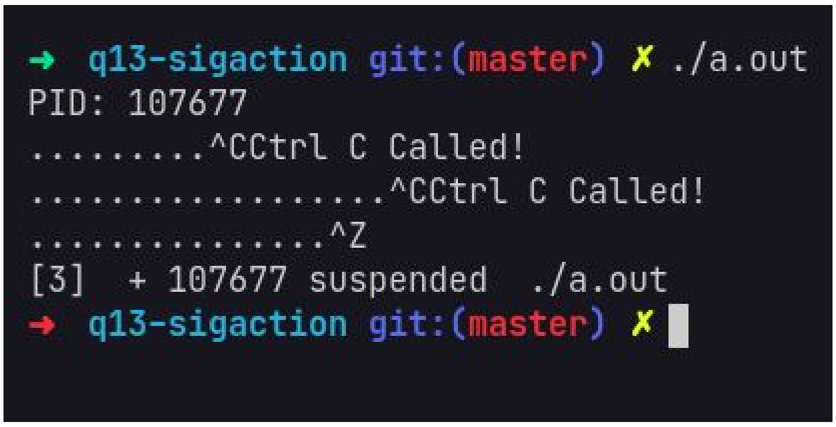


```
➜  q12-brk-sbrk git:(master) ✗ ./a.out
➜  q12-brk-sbrk git:(master) ✗ 
```

with brk()

rt_sigaction()

```c
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

void handler(int num) {
  char *buf = "Ctrl C Called!\n";
  int len = strlen(buf);
  write(STDOUT_FILENO, buf, len);
}

int main(int argc, char const *argv[]) {
  struct sigaction sa;
  sa.sa_handler = handler;
  sigaction(SIGINT, &sa, NULL);

  printf("PID: %d", getpid());
  while (1) {
    printf("...");
    fflush(stdout);
    sleep(1);
  }
  return 0;
}
```

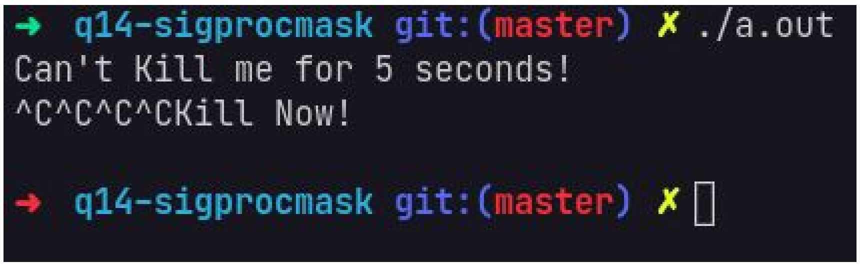


```
➜  q13-sigaction git:(master) ✗ ./a.out
PID: 107677
.........^CCtrl C Called!
..................^CCtrl C Called!
................^Z
[3]  + 107677 suspended  ./a.out
➜  q13-sigaction git:(master) ✗
```

program does not stop after pressing Ctrl C

rt_sigprocmask()

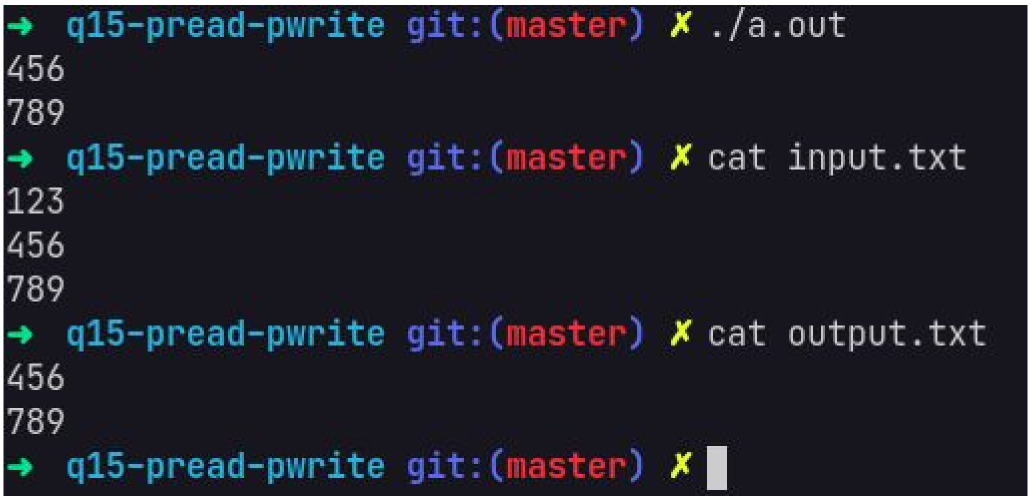
```c
#include <signal.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char const *argv[]) {
  struct sigaction act, oact;

  sigset_t new_mask;
  sigset_t old_mask;

  /* initialize the new signal mask */
  sigfillset(&new_mask);

  /* block all signals */
  sigprocmask(SIG_SETMASK, &new_mask, &old_mask);

  printf("Can't Kill me for 5 seconds!\n");
  fflush(stdout);
  sleep(5);
  printf("Kill Now!\n");

  /* restore signal mask */
  sigprocmask(SIG_SETMASK, &old_mask, NULL);
  return 0;
}
```



```
➜  q14-sigprocmask git:(master) ✗ ./a.out
Can't Kill me for 5 seconds!
^C^C^C^CKill Now!

➜  q14-sigprocmask git:(master) ✗
```

pread() pwrite()

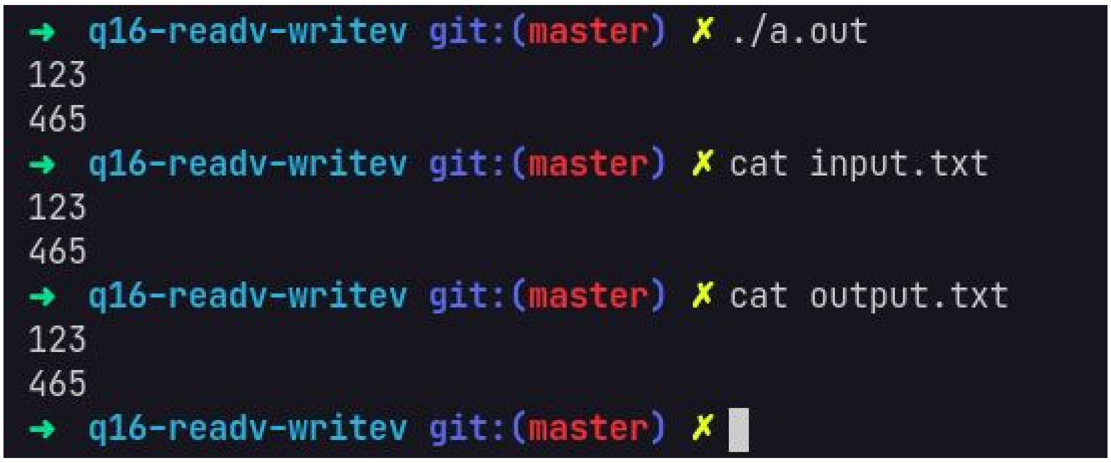
```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>

#define OFFSET 4

int main(int argc, char const *argv[]) {
  int fd1 = open("./input.txt", O_RDONLY);
  int fd2 = open("./output.txt", O_WRONLY);

  if (fd1 < 0 || fd2 < 0) {
    printf("Cannot open files\n");
    exit(1);
  }

  struct stat st;
  fstat(fd1, &st);
  int size = st.st_size;

  char *c = (char *)calloc(size, sizeof(char));

  pread(fd1, c, size, OFFSET);
  printf("%s", c);
  pwrite(fd2, c, size - OFFSET, 0);

  close(fd1);
  close(fd2);
  free(c);
  return 0;
}
```



```
➜  q15-pread-pwrite git:(master) ✗ ./a.out
456
789
➜  q15-pread-pwrite git:(master) ✗ cat input.txt
123
456
789
➜  q15-pread-pwrite git:(master) ✗ cat output.txt
456
789
➜  q15-pread-pwrite git:(master) ✗
```

readv() and writev()

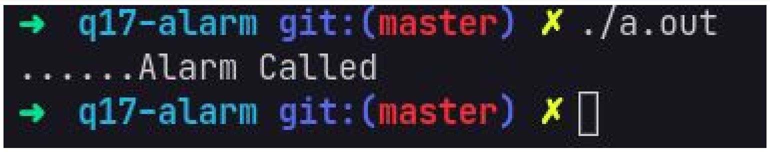
```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>

#define VECTOR_COUNT 3

int main(int argc, char const *argv[]) {
  int fd1 = open("input.txt", O_RDONLY);
  int fd2 = open("output.txt", O_WRONLY);

  struct iovec vec[VECTOR_COUNT];

  struct stat st;
  fstat(fd1, &st);
  int size = st.st_size;

  for (int i = 0; i < VECTOR_COUNT - 1; ++i) {
    vec[i].iov_base = (char *)calloc(size / VECTOR_COUNT, sizeof(char));
    vec[i].iov_len = size / VECTOR_COUNT;
  }
  int rem = (size - (size / VECTOR_COUNT) * (VECTOR_COUNT - 1));
  vec[VECTOR_COUNT - 1].iov_base = (char *)calloc(rem, sizeof(char));
  vec[VECTOR_COUNT - 1].iov_len = rem;

  readv(fd1, vec, VECTOR_COUNT);
  writev(fd2, vec, VECTOR_COUNT);

  for (int i = 0; i < VECTOR_COUNT; ++i) {
    printf("%s", vec[i].iov_base);
  }

  return 0;
}
```



```
➜  q16-readv-writev git:(master) ✗ ./a.out
123
465
➜  q16-readv-writev git:(master) ✗ cat input.txt
123
465
➜  q16-readv-writev git:(master) ✗ cat output.txt
123
465
➜  q16-readv-writev git:(master) ✗
```

alarm()

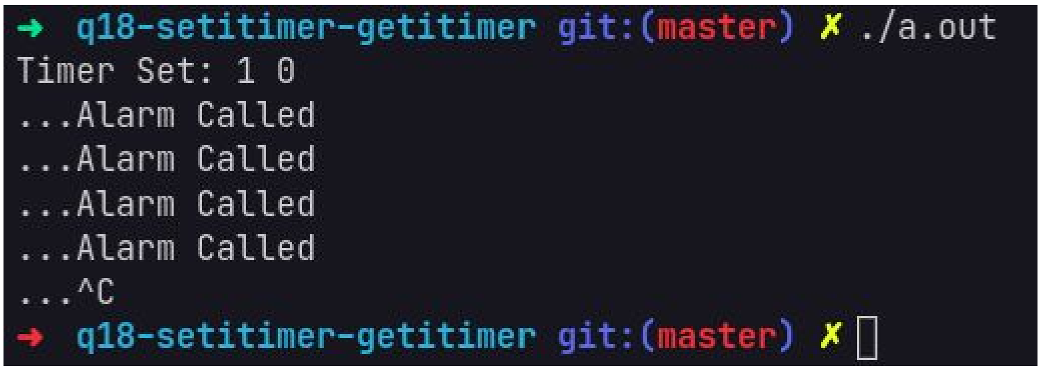
```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>

void sig_handler(int signum) {
  printf("Alarm Called\n");
  exit(0);
}

int main() {

  signal(SIGALRM, sig_handler);

  alarm(2);

  while (1) {
    printf("...");
    fflush(stdout);
    sleep(1);
  }
  return 0;
}
```



```
➜  q17-alarm git:(master) ✗ ./a.out
......Alarm Called
➜  q17-alarm git:(master) ✗ 
```

getitimer() setitimer()

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>

void sig_handler(int signum) { printf("Alarm Called\n"); }

int main() {

  struct itimerval t, test;

  t.it_interval.tv_sec = 1;
  t.it_interval.tv_usec = 0;
  t.it_value.tv_sec = 1;
  t.it_value.tv_usec = 0;

  signal(SIGALRM, sig_handler);
  setitimer(ITIMER_REAL, &t, NULL);
  getitimer(ITIMER_REAL, &test);

  printf("Timer Set: %d %d\n", test.it_interval.tv_sec,
test.it_value.tv_sec);

  while (1) {
    printf("...");
    fflush(stdout);
    sleep(1);
  }
  return 0;
}
```



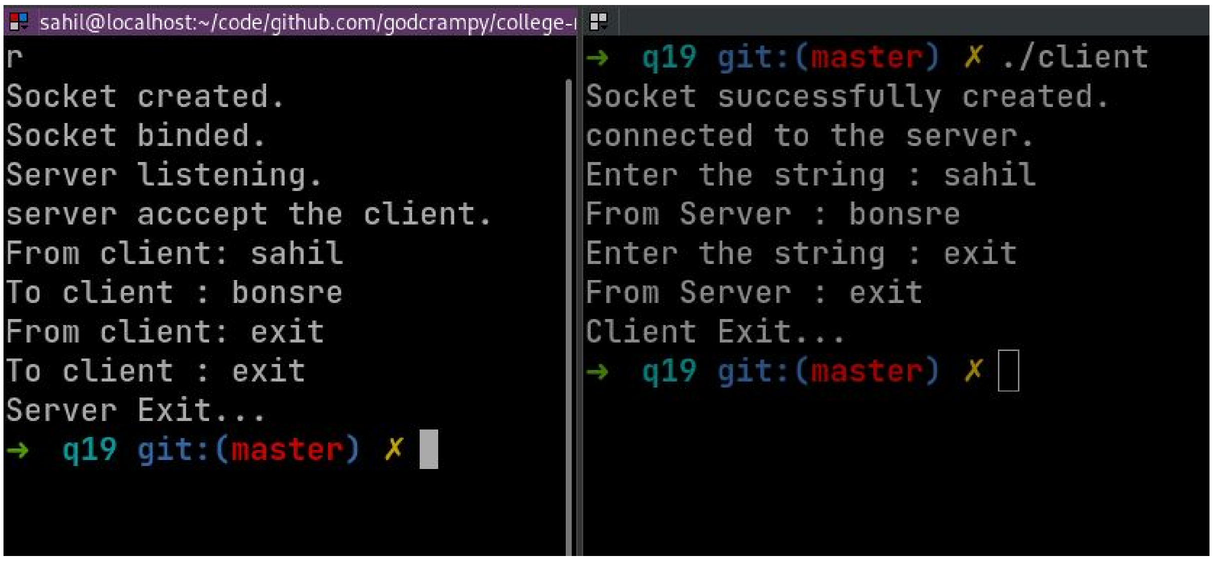
```
➜ q18-setitimer-getitimer git:(master) ✗ ./a.out
Timer Set: 1 0
...Alarm Called
...Alarm Called
...Alarm Called
...Alarm Called
...^C
➜ q18-setitimer-getitimer git:(master) ✗ 
```

socket(), connect(), accept(), sendto(), recvfrom(), sendmsg(), recvmsg(), shutdown(), bind(), listen(), getsocketname(), exit()

```c
#include <fcntl.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define MAX 80
#define PORT 8080
#define SA struct sockaddr

void communicate(int sockfd) {
  char buff[MAX];
  int n;
  // infinite loop for chat
  for (;;) {
    bzero(buff, MAX);

    // read the message from client and copy it in buffer
    read(sockfd, buff, sizeof(buff));
    // print buffer which contains the client contents
    printf("From client: %sTo client : ", buff);
    bzero(buff, MAX);
    n = 0;
    // copy server message in the buffer
    while ((buff[n++] = getchar()) != '\n')
      ;

    // and send that buffer to client
    write(sockfd, buff, sizeof(buff));
    // if msg contains "Exit" then server exit and chat ended.
    if (strncmp("exit", buff, 4) == 0) {
      printf("Server Exit...\n");
      break;
    }
  }
}

int main() {
  int sfd, conn_fd, len;
  struct sockaddr_in servaddr, cli;

  // Create Socket
  sfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sfd == -1) {
    printf("socket creation failed.\n");
    exit(0);
  } else {
    printf("Socket created.\n");
  }
  bzero(&servaddr, sizeof(servaddr));

  // assign IP, PORT
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servaddr.sin_port = htons(PORT);

  // Binding socket to given IP
  if ((bind(sfd, (SA *)&servaddr, sizeof(servaddr))) != 0) {
    printf("socket bind failed.\n");
    exit(0);
  } else {
    printf("Socket binded.\n");
  }

  // Now server is ready to listen
  if ((listen(sfd, 5)) != 0) {
    printf("Listen failed.\n");
    exit(0);
  } else
    printf("Server listening.\n");
  len = sizeof(cli);

  // Accept the data packet from client
  conn_fd = accept(sfd, (SA *)&cli, &len);
  if (conn_fd < 0) {
    printf("server acccept failed.\n");
    exit(0);
  } else
    printf("server acccept the client.\n");

  communicate(conn_fd);

  // close the socket
  close(sfd);
}
```

```
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#define MAX 80
#define PORT 8080
#define SA struct sockaddr

void communicate(int sockfd, struct sockaddr_in servaddr) {
  char buff[MAX];
  int n;
  for (;;) {
    bzero(buff, sizeof(buff));
    printf("Enter the string : ");
    n = 0;
    while ((buff[n++] = getchar()) != '\n')
      ;
    // write(sockfd, buff, sizeof(buff));
    sendto(sockfd, buff, sizeof(buff), 0, (SA *)&servaddr,
sizeof(servaddr)),
        bzero(buff, sizeof(buff));
    // read(sockfd, buff, sizeof(buff));
    int size = sizeof(servaddr);
    recvfrom(sockfd, buff, sizeof(buff), 0, (SA *)&servaddr, &size);
    printf("From Server : %s", buff);
    if ((strncmp(buff, "exit", 4)) == 0) {
      printf("Client Exit...\n");
      break;
    }
```

```c
  }
}

int main() {
  int sockfd, connfd;
  struct sockaddr_in servaddr, cli;

  // socket create
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd == -1) {
    printf("socket creation failed.\n");
    exit(0);
  } else
    printf("Socket successfully created.\n");
  bzero(&servaddr, sizeof(servaddr));

  // assign IP, PORT
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
  servaddr.sin_port = htons(PORT);

  // connect the client socket to server socket
  if (connect(sockfd, (SA *)&servaddr, sizeof(servaddr)) != 0) {
    printf("connection with the server failed.\n");
    exit(0);
  } else
    printf("connected to the server.\n");

  // function for chat
  communicate(sockfd, servaddr);

  // close the socket
  close(sockfd);
}
```



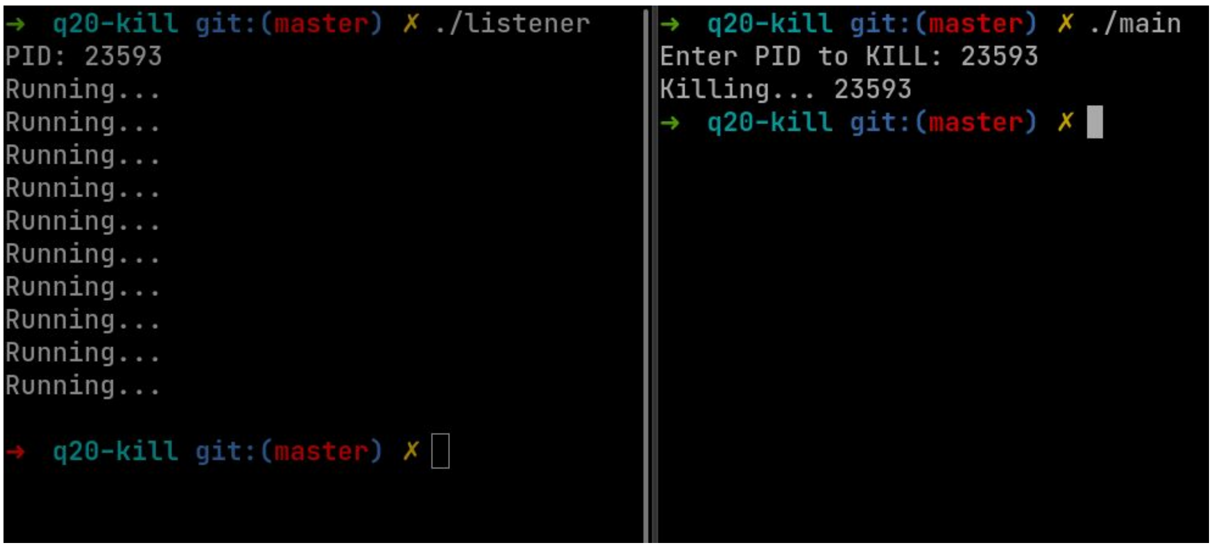
```
sahil@localhost:~/code/github.com/godcrampy/college-
r
Socket created.
Socket binded.
Server listening.
server acccept the client.
From client: sahil
To client : bonsre
From client: exit
To client : exit
Server Exit...
→  q19 git:(master) ✗
```

```
→  q19 git:(master) ✗ ./client
Socket successfully created.
connected to the server.
Enter the string : sahil
From Server : bonsre
Enter the string : exit
From Server : exit
Client Exit...
→  q19 git:(master) ✗
```

kill()

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char const *argv[]) {
  int pid = getpid();
  printf("PID: %d\n", pid);
  while (1) {
    printf("Running...\n");
    fflush(stdout);
    sleep(1);
  }
  return 0;
}
```

```c
#include <signal.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char const *argv[]) {
  int pid;
  printf("Enter PID to KILL: ");
  scanf("%d", &pid);
  printf("Killing... %d\n", pid);
  kill(pid, SIGINT);
  return 0;
}
```

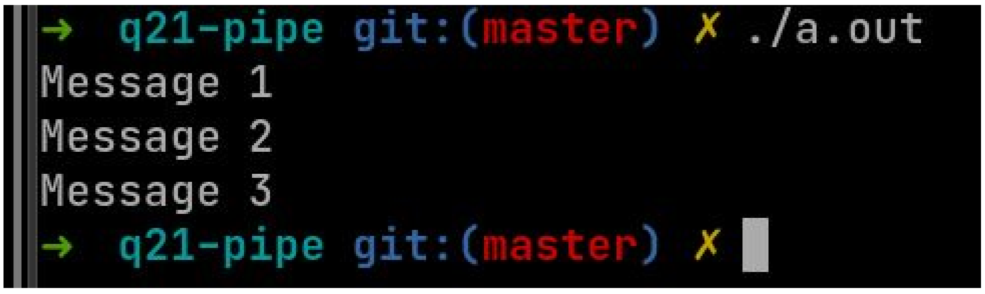


```
→ q20-kill git:(master) ✗ ./listener
PID: 23593
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...

→ q20-kill git:(master) ✗
```

```
→ q20-kill git:(master) ✗ ./main
Enter PID to KILL: 23593
Killing... 23593
→ q20-kill git:(master) ✗
```

pipe()

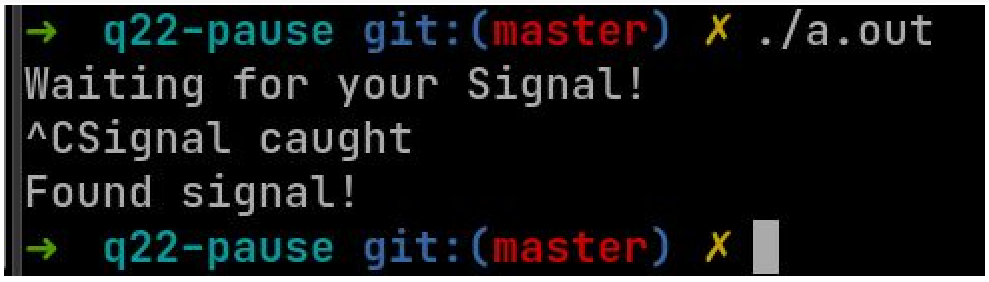
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define MSGSIZE 16
char *msg1 = "Message 1";
char *msg2 = "Message 2";
char *msg3 = "Message 3";

int main() {
  char inbuf[MSGSIZE];
  int p[2], i;

  if (pipe(p) < 0)
    exit(EXIT_FAILURE);

  write(p[1], msg1, MSGSIZE);
  write(p[1], msg2, MSGSIZE);
  write(p[1], msg3, MSGSIZE);

  for (i = 0; i < 3; i++) {
    read(p[0], inbuf, MSGSIZE);
    printf("% s\n", inbuf);
  }
  return 0;
}
```



```
→  q21-pipe git:(master) ✗ ./a.out
Message 1
Message 2
Message 3
→  q21-pipe git:(master) ✗ 
```

pause()

```c
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

static void handler() { printf("Signal caught\n"); }

int main() {
  signal(SIGINT, handler);
  printf("Waiting for your Signal!\n");
  pause();
  printf("Found signal!\n");
  return (0);
}
```



```
→  q22-pause git:(master) ✗ ./a.out
Waiting for your Signal!
^CSignal caught
Found signal!
→  q22-pause git:(master) ✗ 
```