**/\* uniquify.**c **\*/**

int main(int argc, char \*argv[]) {

int pfd[2], sfd[2];

FILE \*stream, \*stream2, \*stream3;

pid\_t status1, status2;

pipe (pfd);pipe (sfd);

// suppressor child process

pid\_t sup\_cid; sup\_cid = fork();

if (sup\_cid == 0) {

// close irrelevant pipe fds

close(pfd[0]);close(pfd[1]);

// handle sfd pipe

close(sfd[1]);dup2(sfd[0],0);

close(sfd[0]); char currWord[37];

char nextWord[37];int wcount = 1;

fgets(currWord, 37, stdin);

while (fgets(nextWord, 37, stdin) != NULL) {

if (strcmp(currWord, nextWord) == 0) {

wcount++; continue;}

printf("%-5d%s", wcount, currWord);

wcount = 1;

strcpy(currWord, nextWord);

memset(nextWord, '\0', 37);}

printf("%-5d%s", wcount, currWord);

exit(0);}

// sort child process

pid\_t sort\_cid; sort\_cid = fork();

if (sort\_cid == 0) {

// handle pfd pipe

close(pfd[1]); dup2(pfd[0], 0);

close(pfd[0]);

// handle sfd pipe

close(sfd[0]);dup2(sfd[1], 1);

close(sfd[1]);

execl("/usr/bin/sort", "sort", (char \*) NULL);exit(1);}

// parent process

char c; int count = 0;

char buff[37]; close(pfd[0]);

stream = fdopen(pfd[1],"w");

while ((c = fgetc(stdin)) != EOF) {

// pick letters only

if (isalpha(c)) {

if (count < 35) {

buff[count++] = tolower(c);}

} else {if (count >= 5) {

// store in buff

buff[count++] = '\n';

buff[count++] = '\0';

fputs(buff, stream);}

memset(&buff, '\0', 37);

count = 0;}}fclose(stream);

close(pfd[1]);close(sfd[0]);

close(sfd[1]);wait(&status1);

wait(&status2);return 0;}

**/\* spawn.c\*/**

char parent[15];

char\*child[]={"myname=john","myname=mary"}; int main(int argc, char \*argv[], char \*envp[]){

int i,pid[2],whom,esize,status;

char \*\*newenvp;

sprintf(parent,"parent=%d",getpid());

for(esize=0; envp[esize] != NULL; esize++);

newenvp= malloc((esize+2)\*sizeof(char \*));newenvp[0]=parent;

for (i=0; i<=esize; i++)

newenvp[i+2]=envp[i];

for (i=0; i<2; i++) {

pid[i]=fork(); if (pid[i] == 0) {

newenvp[1]=child[i]; sleep(4);

/\* we will redirect stdout \*/

close(1);

open((i ? "mary" : "john"),O\_WRONLY |O\_CREAT |O\_TRUNC,0644);

execle("prenv","prenv",NULL,newenvp);

exit(4\*(i+1));}}for (i=0; i<=1; i++ ) {

whom=wait(&status);

if (whom == pid[0] )

printf("John died with status=%x\n",status);

else if (whom == pid[1] )

printf("Alas poor Mary died with status=%x\n",status);}}

**/\*midtermq1.c \*/**

int main(int argc, char \*\*argv) {

DIR \*dirp;struct dirent \*direntP;

struct stat buff;FILE \*stream;

int pfd[2],pid,status;

pipe(pfd);pid = fork();

if (pid == 0) {close(pfd[1]);

dup2(pfd[0],0);close(pfd[0]);

execl("/usr/bin/sort", "sort", (char \*) NULL);}

close(pfd[0]);dup2(pfd[1], 1);

close(pfd[1]);dirp = opendir(argv[1]);

while ((direntP=readdir(dirp)) != NULL) {

stat(direntP->d\_name, &buff);

if (S\_ISREG(buff.st\_mode)) {

printf("%s\n", direntP->d\_name);}}

fclose(stdout); wait(&status);}}

//DIRS S\_ISREG(buf.st\_mode),

S\_ISDIR(buf.st\_mode),

S\_ISBLK(buf.st\_mode),

S\_ISCHR(buf.st\_mode),

S\_ISLNK(buf.st\_mode)

**/\*maryshare.c & johnshare.c \*/**

int sid,\*array,j,sum=0;

sid=shmget(KEY,100\*sizeof (int),IPC\_CREAT |0660);

array=((int \*) shmat(sid,NULL,0));

for (j=0;j<=100;j++) sum+=array[j];

printf("Mary says array sum is %d\n",sum);

// john without the following

shmdt((char \*) array);

shmctl(sid,IPC\_RMID,0);

**/\*share.h \*/**

#define MKEY 77700 QKEY 88800 PERFECT\_MSG 2 REGISTER\_MSG 1

PROCESS\_INDEX\_MSG 3 BITSIZE 1050000

typedef struct process\_struct{

pid\_t pid;

int numberOfPerfect;

int numberOfTested;

int numberOfNotTested;

} process\_struct;

typedef struct sharedMemory\_struct{

// 0 untested; 1 tested

int bitmap[BITSIZE];

int perfectNumsFound[20];

process\_struct processes[20];

int manager\_pid;

} sharedMemory\_struct;

typedef struct message\_struct{

long msg\_type; int content;

} message\_struct;

**/\*manage.c \*/**

sharedMemory\_struct \*shmem;

int shmID,msgID,idxOfProcess;

void terminateAll(int sig){

int i;int j; for (i=0;i<20;i++)

if (shmem->processes[i].pid)

kill(shmem->processes[i].pid, SIGINT);

sleep(5); shmdt(shmem);

shmctl(shmID, IPC\_RMID, NULL);

msgctl(msgID, IPC\_RMID, NULL);

exit(0);}

int main(int argc, char \*argv[]){

//shared memory

shmID = shmget(MKEY, sizeof(sharedMemory\_struct), 0666|IPC\_EXCL|IPC\_CREAT);

shmem = shmat(shmID, NULL, 0);

memset(shmem->bitmap, '\0', sizeof(shmem));

shmem->manager\_pid = getpid();

// signal handler

struct sigaction act;

act.sa\_handler = terminateAll;

sigaction(SIGINT, &act, NULL);

sigaction(SIGHUP, &act, NULL);

sigaction(SIGQUIT, &act, NULL);

//message queue

msgID = msgget(QKEY, 0666|IPC\_CREAT);

message\_struct \*msg\_buffer;

msg\_buffer = malloc(sizeof(message\_struct));

while (1) {

memset(msg\_buffer, '\0', sizeof(message\_struct));

msgrcv(msgID, msg\_buffer, sizeof(msg\_buffer->content), -2, 0);

if (msg\_buffer->msg\_type == REGISTER\_MSG) {

for (j=0;j<20;j++) {

if(shmem->processes[j].pid == 0) {

shmem->processes[j].pid = msg\_buffer->content;break;}}

msg\_buffer->msg\_type = PROCESS\_INDEX\_MSG;

msg\_buffer->content = j;

msgsnd(msgID, msg\_buffer, sizeof(msg\_buffer->content), 0);}

else if (msg\_buffer->msg\_type == PERFECT\_MSG) {

for (j=0;j<20;j++) {

if (shmem->perfectNumsFound[j] == msg\_buffer->content) break;

if(shmem->perfectNumsFound[j] == 0) {

shmem->perfectNumsFound[j] = msg\_buffer->content; break;}}}}}

**/\*compute.c \*/**

int isPerfect(int curr) {

if (curr < 2) {return 0;}

int sum = 1, i;

for (i=2;i<curr/2+1;i++)

if (!(curr%i)) sum+=i;

return sum == curr;}

void terminate(int sig){

memset(&shmem->processes[idxOfProcess], 0, sizeof(process\_struct));

shmdt(shmem);exit(0);}

int main(int argc, char\* argv[]) {

int start;start = atoi(argv[1]);

//TODO: sighandler+shmem+msgQueue

msg\_buffer->msg\_type = REGISTER\_MSG;

msg\_buffer->content = getpid();

int nread = msgsnd(msgID,msg\_buffer,sizeof(msg\_buffer->content),0);

msgrcv(msgID, msg\_buffer, sizeof(msg\_buffer->content), PROCESS\_INDEX\_MSG, 0);

idxOfProcess = msg\_buffer->content;

int curr = start;

while (curr < BITSIZE\*32) {

int segIdx = (curr-2)/32;

int bitIdx = (curr-2)%32;

if (!(shmem->bitmap[segIdx]&(1<<bitIdx))) {

if(isPerfect(curr)) {

msg\_buffer->msg\_type = PERFECT\_MSG;

msg\_buffer->content = curr;

msgsnd(msgID, msg\_buffer, sizeof(msg\_buffer->content),0);

shmem->processes[idxOfProcess].numberOfPerfect++;}shmem->processes[idxOfProcess].numberOfTested++; shmem->bitmap[segIdx] |= (1<<bitIdx); //tested} else {shmem->processes[idxOfProcess].numberOfNotTested++;}curr++;}}

**/\*report.c -k \*/**

if (strcmp(argv[1],"-k")==0)

kill(shmem->manager\_pid, SIGINT);

**/\*oldsignals.c \*/**

jmp\_buf jmpenv; int n;

int main() {int begin;

void status();void query();

if (setjmp(jmpenv)) {

printf("Starting (0 to end): ");

scanf("%d",&begin);

if (begin==0) exit(0);}

else begin=2; signal(SIGINT, status);

signal(SIGALRM, status);

signal(SIGQUIT, query);

alarm(20); perfect(begin);}

void perfect(int start){

int i,sum; n=start; while (1) {sum=1;

for (i=2;i<n;i++) if (!(n%i)) sum+=i;

if (sum==n) printf("%d is perfect\n",n);

n++;}} void status(int signum)

{signal(signum, SIG\_IGN);//ignored

alarm(0); // cancel

if (signum == SIGINT) printf("Interrupt");

if (signum == SIGALRM) printf("Timer");

printf("processing %d\n",n);

alarm(20);/\*restart\*/

signal(signum, status);}//1:ret setjmp

void query() {longjmp(jmpenv,1);}

**/\*chdate.c \*/**

struct stat statbuf; struct utimbuf newt;

stat(argv[1],&statbuf);

printf("Lastaccessed: %s\n",ctime(&statbuf.st\_atime));

printf("Lastmodified: %s\n",ctime(&statbuf.st\_mtim));printf("Last changed: %s\n",ctime(&statbuf.st\_ctime;

newt.actime = statbuf.st\_atime -600;

newt.modtime= statbuf.st\_mtime +600;

utime(argv[1],&newt);

**/\*copy.c buffer\*/**

int fd1,fd2;char buff[4096];

int size, iosize; fd1=open(argv[1],0);

fd2=open(argv[2],O\_WRONLY | O\_CREAT |O\_TRUNC,0644);

while ((iosize=read(fd1,buff,size)) >0 )

write(fd2,buff,iosize);

**/\*copy.c stdio\*/**

FILE \*fd1,\*fd2; int c;

fd1=fopen(argv[1],"r");

fd2=fopen(argv[2],"w");

while ((c=getc(fd1))!=EOF)

putc(c,fd2);fclose(fd2);

**/\*myar.c \*/**

typedef struct ar\_hdr Header;

int readNext(int fd, Header\* arHeader) {

int nread = read(fd, arHeader, sizeof(Header));

if (nread != sizeof(Header)) {

return -1;}return nread;}

**// main method**

if (access( archiveFile, F\_OK) != -1) {

fd = open(archiveFile, O\_APPEND | O\_RDWR);

char magic[9];magic[8] = '\0';

read(fd, magic, 8);

if(strcmp(magic, ARMAG) != 0){

printf("Error: File not recognized\n");

close(fd); exit(-1);}

} else {

if (option == 'q' || option == 'A') {

fd = open(archiveFile, O\_RDWR|O\_CREAT, 0666);

write(fd, ARMAG, 8);

**// switch use**

switch(option) {

case 'q':

appendFiles(fd, files, argc-3);

break; default:

printf("wrong option %c", option);

// show files

void showfiles(int fd) {

Header \*arHeader = malloc(sizeof(Header));

char buff[16]; char\* ptr;

while(readNext(fd, arHeader) == sizeof(Header)) {

printf("%s\n", rHeader->ar\_name);

int sizeOfContent = atoi(arHeader->ar\_size);

if (sizeOfContent%2==1) sizeOfContent++;

lseek(fd, sizeOfContent, SEEK\_CUR);}

free(arHeader);}

**// append**

int append(int fd, char\* file) {

struct stat\* fileInfo = (struct stat\*) malloc(sizeof(struct stat));

Header\* arHeader = (Header\*) malloc(sizeof(Header));

stat(file, fileInfo);

char filename[16];

strcpy(filename, file);

filename[strlen(file)] = '/';

strcpy(arHeader->ar\_name, filename);

sprintf(arHeader->ar\_date, "%-12ld", fileInfo->st\_mtime);

sprintf(arHeader->ar\_uid,"%-6u", fileInfo->st\_uid);

sprintf(arHeader->ar\_gid,"%-6u", fileInfo->st\_gid);

sprintf(arHeader->ar\_mode,"%-8o",fileInfo->st\_mode);

sprintf(arHeader->ar\_size,"%-10lld",fileInfo->st\_size);

sprintf(arHeader->ar\_fmag,"%-2s",ARFMAG);

int n\_read = write(fd, arHeader, sizeof(Header));

// write content

int fd\_file = open(file, O\_RDONLY);

int sizeOfContent = (int)fileInfo->st\_blksize;

char fbuff[sizeOfContent];

while ((n\_read=read(fd\_file, fbuff, sizeOfContent)) > 0) {

write(fd, fbuff, n\_read);

if ((lseek(fd, 0, SEEK\_END) % 2) == 1) write(fd, "\n", 1);}free(fileInfo);

free(arHeader);return 0;}

**// appendAll**

void appendAll(int fd, char\* selfName) {

DIR \*dir; struct dirent \*direntP;

dir = opendir (".")

while ((direntP = readdir(dir)) != NULL) {

if (direntP->d\_type == DT\_REG) {

struct stat \*fstat = (struct stat\*) malloc(sizeof(struct stat));

stat(direntP->d\_name, fstat);

if ( difftime(time(NULL), fstat->st\_mtime) <= 7200 ) {

if ((strncmp(direntP->d\_name, selfName, strlen(selfName))) != 0) {

append(fd, direntP->d\_name);}}}}}

**/\*johnfi.c \*/**

int fifofd,j,n=atoi(argv[1]);

mkfifo("/tmp/john", 0666);

while ((fifofd=open("/tmp/john",O\_WRONLY |O\_NONBLOCK)) == -1) {

if (errno == ENXIO) {sleep(20);}

for (j=1;j<=n;j++)

while (write(fifofd, &j,sizeof (int))== -1) {

sleep(20);} close(fifofd);

printf("I wrote %d number to Mary\n",n);}

**/\*maryfo.c \*/**

int fifofd,j=1,sum,n;

mkfifo("/tmp/john",0666);

fifofd=open("/tmp/john",O\_RDONLY |O\_NONBLOCK);

while (read(fifofd,&sum,sizeof (int))==0) {sleep(20);}

fcntl(fifofd,F\_SETFL,O\_RDONLY);

while (read(fifofd, &n,sizeof (int))!= 0){

sum+=n; j++;}close(fifofd); printf("I Mary read %d numbers totaling %d\n",j,sum);