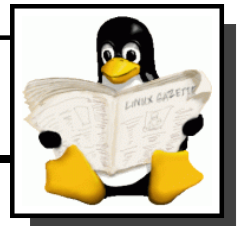
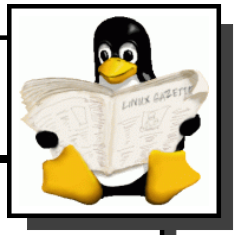
SETTING UP A PIPE



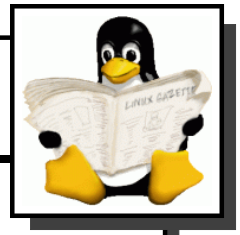
```
#include <stdio.h>
#include <string.h>
#define READ 0
#define WRITE 1
int main(void)
{
     int p[2], pid;
     char line[81];
     FILE *fp;

     if (pipe(p) != 0) {
          fprintf(stderr, "Error Creating Pipe\n");
          exit(1);
     }
```

CHILD PROCESS



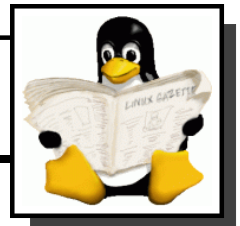
```
if ((pid = fork()) == 0) {
        close(p[WRITE]);

        while (read(p[READ], line, 80) > 0) {
                printf("Child: %s", line);
        }

        close(p[READ]);
}
```

PARENT PROCESS



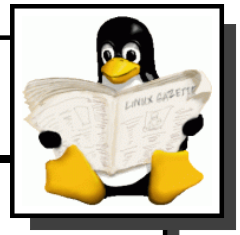
```
   else {
      close(p[READ]);
      if ((fp = fopen("pipe-1.c", "r")) == NULL) {
           fprintf(stderr, "Error reading pipe\n");
           exit(1);
      }
      while (fgets(line, 81, fp) != NULL) {
           write(p[WRITE], line, 80);
      }
      close(p[WRITE]);

      waitpid(pid, NULL, 0);
   }
   exit(0);
}
```

PIPE ACROSS AN EXEC CALL



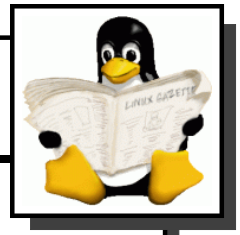
```
< The Usual #includes come here >
int main(int argc, char *argv[])
{
      int i, p[2], pid, lineno = 1, nchars;
      char line[81];

      line[0] = '\0';
      if (argc != 2) {
            fprintf(stderr, "Usage: tunnel <p1>\n");
            exit(1);
      }
      if (pipe(p) != 0) {
            fprintf(stderr, "Error Creating Pipe\n");
            exit(1);
      }
```

CHILD PROCESS



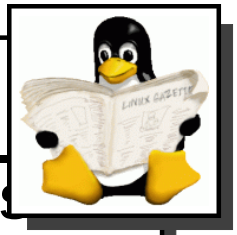
```
if ((pid = fork()) == 0) {
      close(p[READ]);

      if (dup2(p[WRITE], 1) < 0) {
            fprintf(stderr, "Failed tunnel\n");
            exit(2);
      }
      execlp(argv[1], argv[1], NULL);
      fprintf(stderr, "Error running %s\n",
                  argv[1]);
} else {
```

PARENT PROCESS



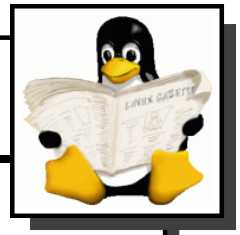
```
    close(p[WRITE]);
    printf("%d: ", lineno++);
    while ((nchars=read(p[READ], line, 80))>0) {
         for (i=0; i<nchars; i++) {
            if (line[i] != '\n') {
               printf("%c", line[i]);
            } else {
               printf("\n%d: ", lineno++);
            }
         }
    }
    printf("\n");
    close(p[READ]);
    waitpid(pid, NULL, 0);
}
```

PIPE ACROSS TWO EXEC CALLS



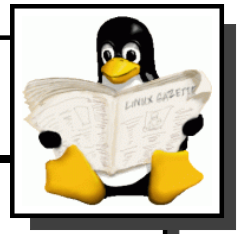
```
< Initialisation is same as in prev. examples
< This is child process >
     if ((pid = fork()) == 0) {
          close(p[READ]);

          if (dup2(p[WRITE], 1) < 0) {
               fprintf(stderr, "Failed tunnel\n");
               exit(2);
          }
          execlp(argv[1], argv[1], NULL);
          fprintf(stderr, "Error running %s\n",
                    argv[1]);
     } else {
```

PARENT PROCESS



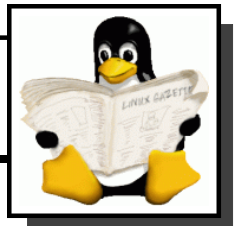
```
        close(p[WRITE]);

        if (dup2(p[READ], 0) < 0) {
           fprintf(stderr, "Failed parent tunnel\n");
           exit(3);
        }

        execlp(argv[2], argv[2], NULL);
        fprintf(stderr, "Error running %s\n",
                argv[2]);
}
```

POPEN CALLS



```
if ((fp1 = popen(argv[1], "r")) == NULL) {
    fprintf(stderr, "popen failed for read\n");
    exit(1);
}

if ((fp2 = popen(argv[2], "w")) == NULL) {
    fprintf(stderr, "popen failed for write\n");
    exit(1);
}
while (fgets(line, 81, fp1) != NULL) {
    fprintf(fp2, "%s", line);
}

pclose(fp1);
pclose(fp2);
```

OUTPUT FROM POPEN PROGRAM



```
[chakcs@archimedes np04]$ p5 ls nl
     1  forkexec
     2  forkexec.c
     3  forkexec.tex
     4  lect_pipes.pdf
     5  lect_pipes.tex
     6  p1
     7  p5
     8  pipe-1.c
     9  pipe-2.c
    10  pipe-3.c
    11  pipe-4.c
    12  pipe-5.c
[chakcs@archimedes np04]$
```