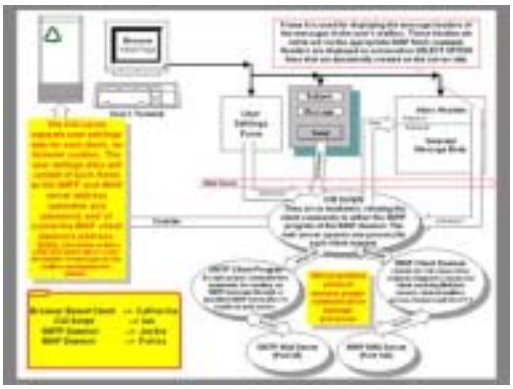
Heriot-Watt University
Edinburgh

Department of Computing & Electrical Engineering

Network Applications



Dr. Hamish Taylor

Group ?	Fotios Bassayiannis

The IMAPD

About the daemon

Imapd is a multi-processed daemon that serves as an IMAP client for multiple concurrent web users. It receives user commands in a proprietary format over an Internet stream socket, translates them into appropriate IMAP commands and performs the required transaction with the IMAP server using another socket.

It all starts with the imapd process first becoming a daemon, by forking, disassociating itself from terminals, closing standard file streams, becoming session leader and of course killing its father.

Imapd then sits and listens on port 4711 for recvcgi connections. Upon accepting a connection, imapd spawns a child process (via a fork() system call) which takes over this specific connection, while the parent imapd process listens to port 4711 for more connection requests that are serviced by more children processes.

Once it takes over, the child answers the recvcgi connection by writing to the returned socket the string: *"OK IMAP daemon client ready (DMIS Rulez!)..."* An appropriate message exchange with the calling recvcgi process follows and the imapd gets and parses the "connection purpose string". Based on the info retrieved from this string, the imapd child process connects to the IMAP server, performs the requested task and then returns appropriate info back to the calling recvcgi process. Once this is done the child dies, and the parent process takes care of its proper burial with the tomb() function (see code for details) so that we don't get memory hungry zombies.

Note that in a concurrent multi-user environment, each child process holds a separate socket to a separate recvcgi process, over the same proprietary port 4711. UNIX takes care of multiplexing the different socket streams over the same port (basically by time-sharing).

The daemon server does not implement a shutdown procedure, triggered by a signal or an IPC message of some sort (like it should). Therefore, if one needs to terminate it, it must be crudely killed.

For interesting comments and explanations, please follow the imapd source code provided below.

Code

```
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stddef.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>

#define SYSTEM_PORT 4711
#define IMAP_PORT   143

char *This_Session[6];       /*will hold values for each IMAP session*/
char *Mail_Buffer;           /*will hold the returned message body or headers*/
FILE *dlog;

/*****************************************************************************/
/*daemon init*/
void init_daemon()
{
  int i;

  if ( fork() != 0 )         /* parent exits */
    exit(0);
  for (i=0; i<3; i++)
  close(i);                 /* close standard streams */

  setsid();                 /* become session leader */
  chdir("$HOME/scratch");   /* change working directory to my scratch */
}

/*****************************************************************************/
/* code to establish a socket */
int create_socket(unsigned short Port)
{
  char   ThisHost[128];
  int    Socket;
  struct sockaddr_in SockAddr;
  struct hostent *Host;

  memset(&SockAddr, 0, sizeof(struct sockaddr_in));   /* clear the address */
  gethostname(ThisHost, 128);               /* put the name of this host in ThisHost*/
  Host = gethostbyname(ThisHost);              /* get ThisHost's address info */

  if (Host == NULL)                            /* ThisHost does not exist !? */
    return(-1);

  SockAddr.sin_family= Host->h_addrtype;              /* this host's address */
  SockAddr.sin_port= htons(Port);            /* the port number for this connection */
  if ((Socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)    /* create socket */
    return(-1);

  if (bind(Socket, &SockAddr, sizeof(struct sockaddr_in)) < 0)
  {
    close(Socket);
    return(-1);                               /* bind address to socket */
  }
  listen(Socket, 5);                         /* max number of queued connects */

  return(Socket);
}
```

```
/*****************************************************************************/
/* wait for a connection to occur on a socket created with establish() */
int accept_connection(int Socket_1)
{ int Socket_2;                                          /* socket of connection */

  if ((Socket_2 = accept(Socket_1, NULL, NULL)) < 0)    /*accept connection*/
    return(-1);                                 /* NULLs are used because we do not wish */
  return(Socket_2);                              /* to know what the IP/port od the
connection */
}                                               /* initiating host */


/*****************************************************************************/
/* as children die we should give them proper burial or else we get
* zombies!  tomb() catches the corpses of the fallen children.
*/
void tomb(void)
{
  int pid;

  while( (pid = waitpid(-1, NULL, WNOHANG)) > 0 )
  {
    fprintf(dlog, "Dead Child pid: %d\n", pid);    /*Request info on ANY child
process*/
    fflush(dlog);

  }                                                   /*Do not store returned info*/
                                    /*Parent Process should not wait for function
return*/
}


/*****************************************************************************/
/*Read the command from the socket*/
void get_command(int Socket, char **Buffer)
{
  int i;

  strcpy(*Buffer, "OK IMAP daemon client ready (DMIS Rulez!)...\n");
  write( Socket, *Buffer, strlen(*Buffer) );

  i = read(Socket, *Buffer, 1024);                        /* read message */
  (*Buffer)[i] = '\0';

  fprintf(dlog, "Command Received: %s", *Buffer);
  fflush(dlog);


  return;
}
```

```
/*****************************************************************************/
/*Parse the received command*/
void parse_command(char **Command)
{
  char *p, *q;
  int i = 0;

  p = q = (char *) malloc(64);

  while (1)
  {
    if (**Command != '#' && **Command != '$')
    {
      *p++ = **Command;
      (*Command)++;
    }
    else if (**Command != '$')
    {
      *p = '\0';
      This_Session[i++] = q;
      p = q = (char *) malloc(64);
      (*Command)++;
    }
    else
    {
      *p = '\0';
      This_Session[i] = q;
      break;
    }

  }

  return;
}


/*****************************************************************************/
/*login to IMAP server and perform requested operation*/
/*commented out print() statements can be used if the server is attached to terminal*/
/*instead of the default of being a daemon                                   */
char *IMAP_operation()
{
  char   *Token, *Helper, *buf;
  int    Socket, i, j, MsgNum;
  struct sockaddr_in SockAddr;
  struct hostent *IMAP_Host;

  char *Buffer  = (char *) malloc(65535);

  /*Setup OutSocket*/
  if ( (Socket = socket(AF_INET, SOCK_STREAM, 0)) < 0 )              /* make socket */
  {
    fprintf( dlog, perror("socket()") );
    fflush(dlog);
  }

  if ( (IMAP_Host = gethostbyname(This_Session[2])) == 0 )           /* get host */
  {
    fprintf( dlog, perror("gethostbyname()") );
    fflush(dlog);
  }


  /*Setup Socket*/
  memset(&SockAddr, 0, sizeof(struct sockaddr_in));
  memcpy(&SockAddr.sin_addr, IMAP_Host->h_addr, IMAP_Host->h_length);   /* set host */
  SockAddr.sin_port = htons(IMAP_PORT);                                 /* set port */
  SockAddr.sin_family = AF_INET;

  /*bind is done automatically by connect*/
  if ( connect( Socket, (struct sockaddr *)&SockAddr, sizeof(SockAddr) ) < 0 )
  {
    fprintf( dlog, perror("connect()") );
    fflush(dlog);
  }

  i = read(Socket, Buffer, 1024);                                   /* read message */
```

```
Buffer[i] = '\0';
fprintf(dlog, "IMAP Message Received = %s\n", Buffer);        /* print message */
fflush(dlog);


if ( !strcmp(This_Session[0], "headers") )
{
  strcpy(Buffer, "1 LOGIN ");  /* load buffer */
  strcat(Buffer, This_Session[3]); /* append username */
  strcat(Buffer, " ");
  strcat(Buffer, This_Session[4]); /* append password */
  strcat(Buffer, "\n");
  write(Socket, Buffer, strlen(Buffer)); /* write data */
  //printf("IMAP Message Sent = %s\n", Buffer);               /* print message */

  i = read(Socket, Buffer, 1024);                             /* read message */
  Buffer[i] = '\0';
  fprintf(dlog, "IMAP Message Received = %s\n", Buffer);     /* print message */
  fflush(dlog);

  strcpy(Buffer, "2 SELECT ");            /* load buffer */
  strcat(Buffer, This_Session[5]);        /* append username */
  strcat(Buffer, "/");
  strcat(Buffer, This_Session[3]);        /* append password */
  strcat(Buffer, "\n");
  write(Socket, Buffer, strlen(Buffer));  /* write data */
  //printf("IMAP Message Sent = %s\n", Buffer);  /* print message */

  i = read(Socket, Buffer, 1024);                               /* read message */
  Buffer[i] = '\0';
  fprintf(dlog, "IMAP Message Received = %s\n", Buffer);        /* print message */
  fflush(dlog);

  Helper = (char *) malloc(65535);
  strcpy(Helper, Buffer);
  Token = strtok(Helper, " ");
  Token = strtok(NULL, " ");

  strcpy(Buffer, "3 FETCH 1:");            /* load buffer */
  //printf("1");

  strcat(Buffer, Token);
  //printf("2");

  strcat(Buffer, " (body[header.fields (from subject date)])\n");
  //printf("3");

  write(Socket, Buffer, strlen(Buffer));          /* write data */
  //printf("4");

  //printf("IMAP Message Sent = %s\n", Buffer);  /* print message */

  j = 0;
  buf = Buffer;

  i = read(Socket, buf, 8);    /*there are always 8 chars to read*/
  j += i;
  buf += i;

  while (j < 65535)              /*read from the stream until specified*/
  {                              /*char sequence*/
    if ( Buffer[j-8] == 'O' &&
         Buffer[j-7] == 'K' &&
         Buffer[j-6] == ' ' &&
         Buffer[j-5] == 'F' &&
         Buffer[j-4] == 'E' &&
         Buffer[j-3] == 'T' &&
         Buffer[j-2] == 'C' &&
         Buffer[j-1] == 'H'    ) break;

    i = read(Socket, buf, 1);        /* read message */
    j+=i;
    buf += i;
  }

  i = read(Socket, buf, 12);  /*read remaining chars after*/
  j+=i;                       /*char sequence*/
```

```
  Buffer[j] = '\0';

  //printf("IMAP Message Received = %s\n", Buffer);  /* print message */

  strcpy(Helper, "4 LOGOUT\n");         /* use helper now, because "Buffer holds*/
                                        /*the info we need*/
  //printf("IMAP Message Sent = %s\n", Helper);  /* print message */
  write(Socket, Helper, strlen(Helper));  /* write data */

  i = read(Socket, Helper, 1024);                /* read message */
  Helper[i] = '\0';
  fprintf(dlog, "IMAP Message Received = %s\n", Helper); /* print message */
  fflush(dlog);

}
else if ( !strcmp(This_Session[0], "body") )
{
  strcpy(Buffer, "1 LOGIN ");                  /* load buffer */
  strcat(Buffer, This_Session[3]);            /* append username */
  strcat(Buffer, " ");
  strcat(Buffer, This_Session[4]);              /* append password */
  strcat(Buffer, "\n");
  write(Socket, Buffer, strlen(Buffer));          /* write data */
  //printf("IMAP Message Sent = %s\n", Buffer);    /* print message */

  i = read(Socket, Buffer, 1024);                   /* read message */
  Buffer[i] = '\0';
  fprintf(dlog, "IMAP Message Received = %s\n", Buffer); /* print message */
  fflush(dlog);

  strcpy(Buffer, "2 SELECT ");                       /* load buffer */
  strcat(Buffer, This_Session[5]);                   /* append username */
  strcat(Buffer, "/");
  strcat(Buffer, This_Session[3]);                  /* append password */
  strcat(Buffer, "\n");
  write(Socket, Buffer, strlen(Buffer));            /* write data */
  //printf("IMAP Message Sent = %s\n", Buffer);      /* print message */

  i = read(Socket, Buffer, 1024);                    /* read message */
  Buffer[i] = '\0';
  fprintf(dlog, "IMAP Message Received = %s\n", Buffer);   /* print message */
  fflush(dlog);

  strcpy(Buffer, "3 FETCH ");                             /* load buffer */
  strcat(Buffer, This_Session[1]);
  strcat(Buffer, " body[TEXT]\n");
  write(Socket, Buffer, strlen(Buffer));                /* write data */
  //printf("IMAP Message Sent = %s\n", Buffer);        /* print message */

  j = 0;
  buf = Buffer;

  i = read(Socket, buf, 8);       /*there are always 8 chars to read*/
  j += i;
  buf += i;

  while (j < 65535)
  {                                /*read until specified char sequence*/
    if ( Buffer[j-8] == 'O' &&
         Buffer[j-7] == 'K' &&
         Buffer[j-6] == ' ' &&
         Buffer[j-5] == 'F' &&
         Buffer[j-4] == 'E' &&
         Buffer[j-3] == 'T' &&
         Buffer[j-2] == 'C' &&
         Buffer[j-1] == 'H'    ) break;
    i = read(Socket, buf, 1);        /* read message */
    j+=i;
    buf += i;
  }

  i = read(Socket, buf, 12);  /*read the filthy remnants*/
  j+=i;
  Buffer[j] = '\0';


  //printf("IMAP Message Received = %s\n", Buffer);  /* print message */
```

```
  Helper = (char *) malloc(1024); /*helper string for temp storage*/

  strcpy(Helper, "4 LOGOUT\n");                  /* load buffer */
  //printf("IMAP Message Sent = %s\n", Helper);  /* print message */
  write(Socket, Helper, strlen(Helper));         /* write data */

  i = read(Socket, Helper, 1024);                  /* read message */
  Helper[i] = '\0';
  fprintf(dlog, "IMAP Message Received = %s\n", Helper);  /* print message */
  fflush(dlog);

}
else if ( !strcmp(This_Session[0], "delete") )
{
  strcpy(Buffer, "1 LOGIN ");              /* load buffer */
  strcat(Buffer, This_Session[3]);         /* append username */
  strcat(Buffer, " ");
  strcat(Buffer, This_Session[4]);         /* append password */
  strcat(Buffer, "\n");
  write(Socket, Buffer, strlen(Buffer));   /* write data */
  //printf("IMAP Message Sent = %s\n", Buffer);     /* print message */

  i = read(Socket, Buffer, 1024);                   /* read message */
  Buffer[i] = '\0';
  fprintf(dlog, "IMAP Message Received = %s\n", Buffer);  /* print message */
  fflush(dlog);

  strcpy(Buffer, "2 SELECT ");            /* load buffer */
  strcat(Buffer, This_Session[5]);        /* append username */
  strcat(Buffer, "/");
  strcat(Buffer, This_Session[3]);        /* append password */
  strcat(Buffer, "\n");
  write(Socket, Buffer, strlen(Buffer));  /* write data */
  //printf("IMAP Message Sent = %s\n", Buffer);  /* print message */

  i = read(Socket, Buffer, 1024);                /* read message */
  Buffer[i] = '\0';
  fprintf(dlog, "IMAP Message Received = %s\n", Buffer);   /* print message */
  fflush(dlog);

  strcpy(Buffer, "3 STORE ");                  /* load buffer */
  strcat(Buffer, This_Session[1]);
  strcat(Buffer, " +FLAGS (\\DELETED)\n");   /*mark msg as deleted*/
  write(Socket, Buffer, strlen(Buffer));      /* write data */
  //printf("IMAP Message Sent = %s\n", Buffer);  /* print message */

  i = read(Socket, Buffer, 2048);                /* read message */
  Buffer[i] = '\0';
  fprintf(dlog, "IMAP Message Received = %s\n", Buffer);  /* print message */
  fflush(dlog);

  Helper = (char *) malloc(1024);

  strcpy(Helper, "4 EXPUNGE\n");                 /* really delete those msgs */
  //printf("IMAP Message Sent = %s\n", Helper);  /* print message */
  write(Socket, Helper, strlen(Helper));         /* write data */

  i = read(Socket, Helper, 1024);               /* read message */
  Helper[i] = '\0';
  fprintf("IMAP Message Received = %s\n", Helper); /* print message */
  fflush(dlog);

  strcpy(Helper, "4 LOGOUT\n");                   /* load buffer */
  //printf("IMAP Message Sent = %s\n", Helper);   /* print message */
  write(Socket, Helper, strlen(Helper));          /* write data */

  i = read(Socket, Helper, 1024);                 /* read message */
  Helper[i] = '\0';
  fprintf(dlog, "IMAP Message Received = %s\n", Helper); /* print message */
  fflush(dlog);

}

close(Socket);

return Buffer;  /*return the important data stored in buffer*/
```

```
}


/*MAIN*************************************************************************/

main()
{
  int Socket_1, Socket_2;
  char *Command, *MailBuffer, *buf;
  int i, j;

  init_daemon();          /*make server a daemon*/

  dlog = fopen("imapd.log", "a");   /*open log file for appending*/

  Command = (char *) malloc(1024);  /*store for command from cgi*/

  if ((Socket_1 = create_socket(SYSTEM_PORT)) < 0)
  {
    fprintf( dlog, perror("create_socket()") );
    fflush(dlog);                /*Write to file right away*/

    exit(1);
  }

  signal(SIGCHLD, tomb);         /*If the parent receives a SIGCHLD from one of its*/
                                 /*children (meaning the child has terminated) then */
                                 /*handle it using the tomb() function*/

  for (;;)
  {


    if ((Socket_2 = accept_connection(Socket_1)) < 0)
    {
      if (errno == EINTR)             /* If accept() was interrupted by a signal, */
        continue;                          /* try again */
      fprintf( dlog, perror("accept()") );    /* else, scream and die*/
      fflush(dlog);

      exit(1);
    }

    switch( fork() )                  /* try to handle connection */
    {
      case -1 :                         /* bad news.  scream and die */
        fprintf( dlog, perror("fork()") );
        fflush(dlog);

        close(Socket_1);
        close(Socket_2);
        exit(1);

      case 0 :                          /* This process is the child, */
        close(Socket_1);                   /*Close listening socket*/
        get_command(Socket_2, &Command);         /* Read the incoming command */
        parse_command(&Command);

        Mail_Buffer = IMAP_operation();

        j = 0;
        i = 1;
        buf = Mail_Buffer;
        while (i > 0)              /* write data to requesting CGI process*/
        {
          i = write(Socket_2, buf, strlen(buf));
          j+=i;
          buf += i;
        }


        close(Socket_2);
        exit(0);

      default :                         /* This process is the parent so, */
        close(Socket_2);                /*Close socket created for reading,*/
```

```
                                        /*as it will be used by the child  */
        continue;                      /* Look for another connection */
    }
  }

  fclose(dlog);    /*when daemon finally terminates, close the log file*/
                   /*Note that this never happens as daemon does not*/
                   /*listen for a termination signal. Therefore it can*/
                   /*only be crudely killed*/
}

/*THE END*************************************************************************/
```