2.11 BSD Sendmail setup guide for UUCP

1. Setting up sendmail on 2.11 BSD

The primary audience for this guide is the PiDP-11 community. This guide provides instructions for setting up 2-way UUCP e-mail between the host Pi and a 2.11 BSD guest system running on a PiDP-11. It is also valid for setting up mail between any 2.11 BSD system running on the simh emulator and its host system, and the first part of this tutorial can probably be used on a real PDP-11. The second part deals with the host end of the setup, and thus is not directly relevant to a real PDP-11, though it will tell you how to configure a modern system to exchange mail with 2.11 BSD.

It is expected that you have already set up UUCP communication between the two systems. A guide for doing so can be found (along with this guide and an upcoming guide for setting up C-news) at:

https://github.com/jwbrase/pdp11-tools/tree/master/howtos/

Setting things up on the 2.11 BSD end involves replacing `/etc/sendmail.cf` with something more suited to the system we're running. All commands in this section of the guide should be run as root. First, backup `sendmail.cf`:

```
    # cp /etc/sendmail.cf /etc/sendmail.cf.bak
```

Next, go to `/usr/src/usr.sbin/sendmail/cf`:

```
    # cd /usr/src/usr.sbin/sendmail/cf
```

This directory is full of template files that can be used for creating a sendmail.cf file. Go with the one called "uucpproto.mc". Since the primary reason that one would want to run 2.11 BSD these days is retrocomputing, we'll be going for a UUCP-only setup. However, because the mail tools on the host system will be designed for a modern environment, they will often automatically place addresses with modern formats into messages (for example: the To: line in a message sent from the host system will generally contain an address in the "`user@host`" format rather than the old UUCP `host!user` format), so we will want sendmail to know what to do with such addresses. Copy `uucpproto.mc` to a file named with the hostname of your system:

```
    # cp uucpproto.mc yourhostname.mc
```

Now, edit `yourhostname.mc` with your favorite editor. Any line beginning with `#` is a comment.

Look for the following (the color won't be there, I've added it for clarification):

```
    # domain
    DDUUCP
    CDUUCP
```

The line "`DDUUCP`" **D**efines a macro called "**D**", with the value "**UUCP**".

The line "`CDUUCP`" defines a **C**lass called "**D**", with one member "**UUCP**".

A class is like a macro, except that it can have more than one value/member. There can be both a macro and a class with the same name. Classes are often used in a sendmail configuration to specify a range of values that a certain part of an address might take that would cause a rule to be triggered. Macros are often used to specify a single value that a rule should add to an address.

Single character upper-case macro and class names are free to be defined by the user for any desired purpose (of course, if you're working from a template, as we are, you should use names defined in the template for the purposes defined in the template). Single character lower-case names are used internally by sendmail for specific purposes. Names of more than one character need to be enclosed in `{curly braces}`.

The template we're using uses "D" as the name for a macro and for a class containing the domain name(s) that apply to this system. Sendmail does have its own internal macros for domain names, but this template does not use them. "UUCP" is a special-purpose domain name that indicates that the message will be delivered by UUCP. If your host Pi receives a domain name from your router, I've found that adding that to the "D" class helps with replying to messages from the host Pi, more on that later. It will also help if you ever plan to have your 2.11 BSD instance receive mail by TCP/IP, though I will not cover that in this guide. So, optionally, change the `CDUUCP` line to:

```
CDUUCP yourdomain
```

You may also want to add a line something like the following:

```
Cwlocalhost $w
```

This will allow sendmail to recognize "`localhost`", as well as the system's hostname, as a name for the system.

Next, let's define a macro for the name of the host Pi:

```
DRpiname
```

Now skip down to the comment "Machine dependent part of ruleset zero". Everything after this consists of rewriting rules. Rewriting rules begin with "R", and have the format:

```
Rinaddr    outaddr    comments
```

Each field must be separated by tabs (the leading "R" isn't a field, just something that identifies the line as a rewriting rule). "`inaddr`" specifies a pattern to be checked against each address. If an address matches the pattern, the address will be rewritten to "`outaddr`". "`comments`" is an optional field that you can use to remind yourself what a rewriting rule does.

Our first two rules will be used to strip out (most) mail whose final destination is the local system. We'll add them between the "Machine dependent part of ruleset zero" comment and the existing rules.

They are:

```
        R$*<@$=w.$=D>  $#local$:$1     Deliver locally
        R$*<@$=w>      $#local$:$1     Deliver locally
```

These rules expect an address of the form `user@<host.optionaldomain>`. The template we're using includes() certain files with standard rules in them that rewrite addresses in this format before our rules see them. The `$*` at the beginning matches any text that precedes the <. In other words, it matches any address regardless of the username. Similar constructions (called "metasymbols" in the documentation) are: `$+`, `$–`, `$*`, `$=`, and `$˜`. The definitions for these metasymbols in the documentation are:

$*	Match zero or more tokens
$+	Match one or more tokens
$−	Match exactly one token
$=x	Match any phrase in class x
$˜x	Match any word not in class x

The `$=w` matches the "w" class we defined earlier, so it matches any hostname that refers to the local system. The `$=D` matches the "D" class we set up earlier. All together, with the previous definitions of `$D` and `$w`, this will match any address of the form `anyuser<@thismachine.yourdomain>`, `anyuser<@thismachine.UUCP>`, or `anyuser<@thismachine>`. The only local addresses it doesn't handle are addresses of the form "anyuser" (in other words, just the name of a user account on the local system without any machine or domain name). These addresses will be handled later.

The rule "rewrites" the address to an instruction to sendmail on how to deliver the mail. `$#` specifies a mailer for sendmail to use. A mailer is a way that sendmail knows about for how to deliver mail. Mailers are defined in sendmail.cf, but we don't have to worry about defining them for the purposes of this guide, as our template includes() files that define a few for us. Our two rules use the "`local`" mailer, which sendmail uses to deliver local mail. The `local` mailer delivers mail to whatever username follows the `$:` token, in this case, that is "`$1`". In the "`outaddr`" section of a rewriting rule, `$n` (where n is a number) refers to the text that matched the nth metasymbol in the "`inaddr`" section of the rule. In the case of the rules we've just written, `$1` inserts the text that matched `$*`, which is the username of the address. In short, these rules say "when you see an address that contains this system's hostname, deliver the mail to the local user named in the address".

Now we will write the rules that will forward UUCP mail on to the next hop towards its destination.

The first one is:

```
        R$+<@$-.$+.UUCP>            $2!$1<@$3.UUCP>
```

This time, I have colored each metasymbol in "`outaddr`" to match the corresponding metasymbol in "`inaddr`". This rule rewrites domain-style address of the form `user<@destination.hop2.hop1.UUCP>` to a more UUCP-ish hybrid address of the form destination!user<@hop2.hop1.UUCP> . Each rule keeps on being applied until it no longer matches, so on the next round through, the address will be matched as:

```
R$+                 <@   $-   .    $+         .UUCP>
destination!user    <@   hop2 .    hop1       .UUCP>
```

and will be rewritten as:

```
hop2!destination!user<@hop1.UUCP>
```

At this point, the rule will not match the address again (after “`destination`” matches the `$-`, there will be no more tokens before the “`.UUCP`” to match the `$+`), so the address will then be passed on to our next rule. This rule will actually be the first of the existing rules that was already in our template file:

```
R<@$+.UUCP>:$+       $1!$2          to old format
```

This relates to an old way of writing addresses as `hostname:user`. I’m not sure if this is used by any 2.11 BSD mail software, or by any modern system, or if it’s generated by any of the files our template includes(), so we may be able to delete this one, but just comment it out for now (by adding a # at the beginning of the line) in case it actually is needed. If anyone who was an expert on sendmail back in the day can comment on this, I’d really appreciate it. The next rule (which is the next rule from the original template) we *will* need, for certain:

```
R$+<@$+.UUCP>       $2!$1          to old format
```

This performs the last step of changing the `hop2!hop1!user<@destination.UUCP>` address above into a full bang-path address, which will then look like: `hop1!hop2!destination!user`. The final rule in this section is the last one in the template before the “everything else must be a local name” comment. It is:

```
R$-!$+         $#uucp$@$1$:$2 host!user
```

This is like the delivery rule for local mail, except in addition to the `$:` specifying the username, we have `$@` to specify the hostname that we’re delivering to. This will be the name of an entry in our system’s `/etc/uucp/L.sys` file. This rule looks for an address in the format of a bang path like `hop1!hop2!destination!user`, and uses the “`uucp`” mailer to queue UUCP to deliver the mail to the neighboring system “`hop1`”, with the rest of the bang path, “`hop2!destination!user`”, as the username. Sendmail on hop1 will deliver the message to “`hop2`”, with `destination!user` as the username, then sendmail on hop2 will deliver the message to “`destination`” with “`user`” as the username.

We now have all mail that has a “UUCP” domain on the address processed. The next step is to deliver any mail to the host Pi that doesn’t have a bang path, or the “UUCP” domain tacked on to the end of the address. We will also deliver this remaining mail over UUCP. As 2.11 BSD has networking functionality, we could also deliver the mail by SMTP, but that is beyond the scope of this guide.

The rest of the rules will be new rules between the last of the rules in the previous section and the "everything else must be a local name" comment. To deliver the remaining mail to the Pi, we'll use the following two rules:

```
R$+<@$R.$=D>          $#uucp$@$R$:$1
R$+<@$R>              $#uucp$@$R$:$1
```

The first rule is only needed if you added anything beyond "UUCP" into the "D" class. The "`$=D`" catches the domain name, if there is one. Recall that we previously set the "R" macro to the hostname of the host Pi.

The next rule is optional: if you want to set up the host Pi to forward mail from your 2.11BSD instance to other machines on your LAN you could do something like this:

```
R$*<@$+.$=D>         $#uucp$@$R$:$1@$2.$3
```

This will pass any e-mail with a domain name in the "D" class to the host Pi, with the full e-mail address provided as the username. The mail server on the host Pi would then need to be configured to pass the mail on appropriately, which is beyond the scope of this guide. By replacing the `$=D` with `$*`, you could even have mail for any address on the internet forwarded to the Pi, but configuring the Pi to exchange mail with the internet at large is **far** beyond the scope of this guide, as running a mail server on the modern internet is not for the faint of heart: Russian spammers would **love** to crack your Pi and induct it into their Viagra-spam botnet, so you'd have to take security very seriously. Meanwhile, every legitimate mail server on the Web assumes that any mail coming out of an IP address block that a given ISP uses for residential internet service is from a cracked device that's been inducted into a botnet, so your mail won't be delivered unless you're paying your ISP for a business-class connection. In short, if you don't already have experience running an internet-connected mail server, this project is not where you should learn that skill.

The next rule takes any remaining address with an @ in it and errors out on it:

```
R$*<@$*>        $#error$:Address is not local, UUCP, or LAN, cannot route!
```

We're now all done with our changes to the template. All that remains now is the last rule that was in the original template. It reads:

```
R$+                         $#local$:$1
```

Remember a couple pages up where I talked about local addresses that are just a username? This rule handles those, and delivers them locally. Save the file and exit your editor, then issue the following command to your shell:

```
# m4 yourhostname.mc > /etc/sendmail.cf
```

This generates the actual `sendmail.cf` file. There is also a file, "`sendmail.fc`" that contains the same information in a binary format. If present, it will be used instead of `sendmail.cf`.

Generate it with:

```
# sendmail -bz
```

Then, you can test to make sure the rules are working with:

```
# sendmail -bt
```

You'll get a ">" prompt. Type a ruleset number (the rules we've added are in set zero), and then an e-mail address. Sendmail will then show you how it is processing that address:

```
     > 0 hop1!hop2!destination!user
     rewrite: ruleset  3   input: "hop1" "!" "hop2" "!" "destination"
"!" "user"
     rewrite: ruleset  8   input: "hop1" "!" "hop2" "!" "destination"
"!" "user"
     ...
     rewrite: ruleset  3 returns: "hop2" "!" "destination" "!" "user"
"<" "@" "hop1" "." "UUCP" ">"
     rewrite: ruleset  0 returns: "^V" "uucp" "^W" "hop1" "^X" "hop2"
"!" "destination" "!" "user"
```

In the final output "`^V`" will be the mailer being used, "`^W`" will be the host the mail will be sent to, and "`^X`" will be the username at that host.
If everything checks out, we're ready to set up things on the Pi side.

2. Configuring postfix on the Pi

Now we need to set up the host Pi to actually receive mail from 2.11 BSD. The first thing to do is install the postfix mail server. Log in with your normal user account and run:

```
$ sudo apt-get install postfix
```

Once that is done, edit `/etc/postfix/master.cf` , and search for the string "uucp". If you see a pair of lines that look like this:

```
uucp      unix  -       n       n       -       -       pipe
  flags=Fqhu user=uucp argv=uux -r -n -z -a$sender - $nexthop!rmail
```

Then this part has already been set up for you. If not, add it at the end of the file.

Now, create or edit the file `/etc/postfix/transport`. Add any hostnames that you want postfix to recognize for your 2.11 BSD system, followed by a tab, then "`uucp:uucpname`", where "`uucpname`" is the name that your existing UUCP installation has configured for the 2.11 BSD system. For instance, if your 2.11 BSD system is called "`mississippi`", and the domain name for your LAN is "`mylan`", you might put something like this (next page):

```
     mississippi         uucp:mississippi
     mississippi.mylan   uucp:mississippi
     mississippi.uucp    uucp:mississippi
```

When done, run:

```
     $ sudo postmap /etc/postfix/transport
```

After that, run:

```
     $ postconf -m
```

Copy the output into a text editor window, you'll need it for the next bit. Now edit `/etc/postfix/main.cf`. Add a line at the end:

```
     transport_maps = hash:/etc/postfix/transport
```

If "`hash`" was in the output from `postconf -m`, no modification is needed. If not, replace "`hash`" with something that was in the `postconf -m` output, maybe "`dbm`", if it's there.

Without closing the file, look for a line that says "`relay_domains = ...`". If one exists, add the names for your 2.11 BSD system that you want postfix to recognize. If not, add a line of the form:

```
     relay_domains = mississippi mississippi.mylan mississippi.uucp
```

Save the file, and run the following to have the running postfix instance load your changes:

```
     $ sudo postfix reload
```

Now try sending a few messages back and forth to see if they arrive.

On the Pi:

```
     $ mail mississipi\!user
```

Where "`user`" is the name of your user account. Note the \ before the !, you'll need this because the ! has a special meaning in many modern shells, so it needs to be escaped. You'll be prompted with "`CC:`" and "`Subject:`". For `CC:`, just hit Ctrl-D. Type in a subject line and a message, then end the message with Ctrl-D. The ^D's won't actually be visible, they just tell you where to type Ctrl-D:

```
     CC: ^D
     Subject: Test
     This is a test
     ^D
```

To avoid waiting till the UUCP cron job does its next run, type:

```
     $ sudo uucico -S mississippi
```

Then, on the 2.11 BSD system, run `Mail`. Note the capital M. 2.11 BSD has two different mail clients, “`mail`” and “`Mail`”. “`Mail`” is a lot easier to work with. The account named “`user`” in the default setup that comes with the PiDP-11 operating systems pack has its shell profile set up so that “`mail`” is an alias for “`Mail`” (so that typing “`mail`” actually runs “`Mail`”, and the normal “`mail`” program is “hidden”), but the root account, and any account you’ve created that you haven’t made the requisite shell profile changes for doesn’t have the alias set up, so typing “`mail`” will actually run “`mail`”. On a side note, the “`mail`” command on most modern Unix systems (and thus on the host Pi) is generally a descendant/clone of the “`Mail`” command on 2 BSD.

When you type “`Mail`”, the output you see should be something like this:

```
$ Mail
Mail version 5.5 6/1/90.  Type ? for help.
"/usr/spool/mail/user": 1 message
>   1 user@piname.mydomain Sat Mar 14 00:17  15/507   "Test"
&
```

The “`&`” is Mail’s prompt character. You can type “`?`” for a list of valid commands, or the number of a message to print it:

```
& 1
Message 1:
From somewhere!user@piname.mydomain Sat Mar 14 00:17:28 2020
To: <mississippi!user@piname.mydomain>
Subject: Test
X-Mailer: mail (GNU Mailutils 3.4)
Date: Sat, 14 Mar 2020 01:36:38 -0500 (CDT)
From: You <user@piname.mydomain>

This is a test
```

You’ll then get another `&` prompt. Now type “`R1`” to reply to the sender of message 1:

```
& R1
To: jon@orthanc.brasenet
Subject: Re: Test

Test back
^D
```

When you hit Ctrl-D mail will print “`EOT`” on the line you were on, then another prompt:

```
EOT
&
```

Now hit “`d1`” to delete the message you just replied to, and `q` to exit back to the shell:

```
& d1
& q
$
```

Now, type the following commands on the Pi. Note that where 2.11 BSD `Mail` gives an “`&`” prompt, mail on the Pi will give a “`?`” prompt.

```
$ sudo uucico -S mississippi
$ mail
"/var/mail/user": 1 message 1 new
>N   1 mississippi!user     Sat Mar 14 02:23  14/448   Re: Test
? 1
Return-Path: <user@mississippi>
X-Original-To: user
Delivered-To: user@piname.mydomain
Received: by piname.mydomain (Postfix, from userid 10)
        id 857022C0928; Sat, 14 Mar 2020 02:23:31 -0500 (CDT)
Received: by mississippi.UUCP (5.52.1/5.17)
        id AA01617; Sat, 14 Mar 2020 01:03:29 -0800
Date: Sat, 14 Mar 2020 01:03:29 -0800
From: mississippi!user (You)
Message-Id: <12003140903.AA01617@mississippi.UUCP>
To: user@piname.mydomain
Subject: Re: Test

Test back
? d1
? q
Held 0 messages in /var/mail/user
$
```

If you managed to get the message and reply back and forth, you now have working UUCP mail between your 2.11 BSD system and its host.