株式会社 ITS MORE
2020年4月設立

Digitally signed by 佐藤 豊
Date: 2020.08.12 21:17:28 +09'00'

2020年8月12日 投稿者: SATOXITS

続々々々GShell

開発：さて、今日はUnix的なるものの核心の一つ、pipe に取り組もうと思います。

社長：それは標準入出力と切っても来れない関係にありますね。

開発：pipeにはファイル名がありませんからね。予めオープンされたディスクリプタとしてしか獲得できない。それは普通、shellから標準入出力として、fd0 とかfd1としてプログラムに渡されるわけです。

社長：ディスク上に記録されないので無限長のデータ処理も可能。今はマルチコアの時代なので、pipeをはさんでまさにパイプライン並列な高速処理ができるようになりました。

開発：shell のパイプ記号 | のわかりやすさもあって 、出力と入力のプログラムを並行して起動してパイプでつなげるという方式は大成功を納めたと思います。

社長：半世紀くらい前に読んだ「Unixの四半世紀」という本でも、ベル研の人たちがこれを思いついた時に大喜びしてたシーンが書かれてた記憶があります。

開発：これがベースになって、いわゆるフィルターコマンドという形で各種のコマンド部品が蓄積されて行きましたね。各プロセスは単に入力を加工して出力に出すだけなので、書きやすいし、それを組み合わせるのは外で、通常はshellがやってくれて、いろんな組み合わせができる。

基盤：まあ、cat | sort | uniq -c | sort なんてのは今でもよく使います。grep も -e とか付けるのが面倒なので、grep | grep | grep … なんて継ぎ足していくことが多いですね。性能的にも問題無いし。

開発：ただこの、いろんな処理が一つにまとまっているのが簡便な一方、ちょっとフィルター的に収まらないプログラムを作る時に残念なことになるわけです。

社長：pipe記法の制約の一つは、a | b という記述で、a と b が同時に起動されないければいけないということだと思います。まあ、データを一気に上から下まで流すという使い方を想定すればそれで良いのですが、そうでないこともある。

開発：なので、このパイプ記法がワンセットとして実現している機能をプリミティブにばらして、ユーザがそれを組み合わせられるようにできると面白いだろうなと思うわけです。

＊　＊　＊

基盤：ここで一つ、昨日入荷した桃ちゃんを頂きたいと思います。うーん、なんか香りが弱いですね。

社長：なんにしろ桃は皮ごと食べるのが常道です。ではひとくち。がりっ

基盤：あれ？めっちゃ固いですね。がりがり

開発：桃とは思えないこの食感。普通なら困ってしまうほどのジューシーさがまるで無い。ごりっ

社長：まあ、果物には当たり外れがありますけどね。しかしこれはひどい…

開発：あのスーパーは昔はこんなものを売るような店じゃなかったですが…

＊　＊　＊

開発：さて、今日はちょっと地道な機能を追加していこうと思います。まず現状のまとめ。

開発：まず、which コマンドを拡充しまして、ワイルドカードが使えるようにしました

```
MacMini% gsh
!1! ver
gsh/0.0.6 (2020-0812a)
--I-- Aug 12 12:08:50(0.000025987s)
!2! which ssh* -ls
-rwxr-xr-x  2157868 Aug  5 21:28:36 /Users/ysato/bin/ssh
-rwxr-xr-x  5936448 Feb  3 15:37:37 /usr/local/bin/sshfs
-rwxr-xr-x  2142352 Jul 10 07:27:01 /usr/bin/ssh
-rwxr-xr-x  1817392 Jul 10 07:27:04 /usr/bin/ssh-add
-rwxr-xr-x  1794016 Jul 10 07:26:51 /usr/bin/ssh-agent
-rwxr-xr-x    10658 Nov  9 20:54:44 /usr/bin/ssh-copy-id
-rwxr-xr-x  1884080 Jul 10 07:27:06 /usr/bin/ssh-keygen
-rwxr-xr-x  1886416 Jul 10 07:26:47 /usr/bin/ssh-keyscan
-rwxr-xr-x  2189040 Jul 10 07:27:04 /usr/sbin/sshd
--I-- Aug 12 12:09:04(0.046707834s)
!3! which *mount* -ls
-rwxr-xr-x       80 Apr 10 03:32:31 /usr/local/bin/vboximg-mount
-rwxr-xr-x    37856 Jul 10 07:26:48 /usr/bin/showmount
-rwxr-xr-x    80688 Jul 10 07:27:01 /usr/sbin/automount
-r-xr-xr-x    41616 Jul 10 07:27:11 /sbin/mount
-rwxr-xr-x    37440 Jul 10 07:27:11 /sbin/mount_9p
-rwxr-xr-x   366208 Jul 10 07:26:19 /sbin/mount_acfs
-rwxr-xr-x    36384 Jul 10 07:27:10 /sbin/mount_afp
-rwxr-xr-x    54016 Jul 10 07:26:19 /sbin/mount_apfs
-rwxr-xr-x    37088 Jul 10 07:26:17 /sbin/mount_cd9660
-rwxr-xr-x    50496 Jul 10 07:26:16 /sbin/mount_cddafs
-r-xr-xr-x    31376 Jul 10 07:27:11 /sbin/mount_devfs
-rwxr-xr-x    37552 Jul 10 07:26:17 /sbin/mount_exfat
-r-xr-xr-x    31584 Jul 10 07:27:11 /sbin/mount_fdesc
-rwxr-xr-x    31456 Jul 10 07:26:20 /sbin/mount_ftp
-rwxr-xr-x    38608 Jul 10 07:26:20 /sbin/mount_hfs
-rwxr-xr-x    38672 Jul 10 07:26:15 /sbin/mount_msdos
-rwxr-xr-x    68224 Jul 10 07:27:11 /sbin/mount_nfs
-rwxr-xr-x    32048 Jul 10 07:26:19 /sbin/mount_ntfs
-r-xr-xr-x    71072 Jul 10 07:27:10 /sbin/mount_smbfs
-rwxr-xr-x    85232 Jul 10 07:26:19 /sbin/mount_udf
-rwxr-xr-x    31504 Jul 10 07:27:10 /sbin/mount_webdav
-r-xr-xr-x    37488 Jul 10 07:27:11 /sbin/umount
--I-- Aug 12 12:09:17(0.023346502s)
```

基盤：うーん、これは普通に便利。インストールされてるパッケージとかライブラリも同じようにできると良いですね。

開発：ヒットしたファイルの属性表示には、file コマンドに食わせるとかも良いかもですね。cksum とか wc とかいろいろ有り得る。まあ、xargsでやれるかも知れませんが、shellのビルトイン記法で。

社長：ワイルドカード検索した結果をフルパスでなくてコマンド名だけで出すモードも欲しいですね。その結果をまた、別のコマンドに食わせる。

開発：which は要するにfind の特殊形だと思うんです。逆に、which を表現できるようなビルトインのfindを作ると良いかなと思います。

開発：あとは、リソース使用量の表示について、ビルトインコマンドでも表示できるようにしました。

```
!4! nop
--I-- Aug 12 12:20:12(0.000007965s)
!5! nop
--I-- Aug 12 12:20:13(0.000005684s)
!6! nop
--I-- Aug 12 12:20:14(0.000006317s)
!7! echo %T
Aug 12 12:20:18
--I-- Aug 12 12:20:18(0.000031015s)
!8! hi -l
!0   Aug 12 12:19:59   117.674µs/t 0.000055s/u 0.000081s/s ver
!1   Aug 12 12:20:05 56.445068ms/t 0.003902s/u 0.032071s/s which ssh* -ls
!2   Aug 12 12:20:08 21.822166ms/t 0.002969s/u 0.013093s/s which *mount* -ls
!3   Aug 12 12:20:12    39.725µs/t 0.000023s/u 0.000022s/s nop
!4   Aug 12 12:20:13    35.263µs/t 0.000023s/u 0.000023s/s nop
!5   Aug 12 12:20:14    29.022µs/t 0.000017s/u 0.000013s/s nop
!6   Aug 12 12:20:18    39.205µs/t 0.000027s/u 0.000024s/s echo %T
```

基盤：内部コマンドレベルのオーバヘッドは30マイクロ秒くらいですね。

社長：せっかくnopというコマンドを作るなら、アセンブラが使えると良いかもですね…

開発：ということで、とりあえず現状をアーカイブします。現在1175行。

Gshell_0.0.6-by-SatoxITS ダウンロード

開発：あともう一点。Goのビルドしたネイティブなバイナリの性能が気になっていたのですが、まあ立ち上がりについてはCからのネイティブと変わらないようです。

```
MacMini% gsh
!1! nop
--I-- Aug 12 13:01:52(0.000017037s)
!2! nop
--I-- Aug 12 13:01:55(0.000015136s)
!3! nop
--I-- Aug 12 13:01:57(0.000005150s)
!4! ./hello-c
C-Hello GShell!!
--I-- Aug 12 13:02:02(0.201046437s)
!5! ./hello-c
C-Hello GShell!!
--I-- Aug 12 13:02:03(0.002508251s)
!6! ./hello-c
C-Hello GShell!!
--I-- Aug 12 13:02:05(0.003453331s)
!7! ./hello-go
Go-Hello GShell!
--I-- Aug 12 13:02:11(0.215221390s)
!8! ./hello-go
Go-Hello GShell!
--I-- Aug 12 13:02:13(0.003537527s)
!9! ./hello-go
Go-Hello GShell!
--I-- Aug 12 13:02:16(0.003917119s)
!10! ./hello.go
--I-- /usr/local/go/bin/go {/usr/local/go/bin/go run ././hello.go}
Go-Hello GShell!
--I-- Aug 12 13:02:29(0.664595709s)
!11! ./hello.go
--I-- /usr/local/go/bin/go {/usr/local/go/bin/go run ././hello.go}
Go-Hello GShell!
--I-- Aug 12 13:02:33(0.243025586s)
!12! ./hello.go
--I-- /usr/local/go/bin/go {/usr/local/go/bin/go run ././hello.go}
Go-Hello GShell!
--I-- Aug 12 13:02:35(0.247824638s)
!13! hi -l
!0   Aug 12 13:01:52   127.912µs/t 0.000056s/u 0.000089s/s nop
!1   Aug 12 13:01:55    36.053µs/t 0.000025s/u 0.000023s/s nop
!2   Aug 12 13:01:57    26.831µs/t 0.000016s/u 0.000015s/s nop
!3   Aug 12 13:02:02 201.083079ms/t 0.000224s/u 0.000578s/s ./hello-c
!4   Aug 12 13:02:03  2.534571ms/t 0.000175s/u 0.000539s/s ./hello-c
!5   Aug 12 13:02:05  3.486139ms/t 0.000168s/u 0.000557s/s ./hello-c
!6   Aug 12 13:02:11 215.254937ms/t 0.000209s/u 0.000583s/s ./hello-go
!7   Aug 12 13:02:13  3.567667ms/t 0.000161s/u 0.000548s/s ./hello-go
!8   Aug 12 13:02:16  3.950176ms/t 0.000172s/u 0.000538s/s ./hello-go
!9   Aug 12 13:02:29 664.64063ms/t 0.000230s/u 0.000590s/s ./hello.go
!10  Aug 12 13:02:33 243.070142ms/t 0.000213s/u 0.000577s/s ./hello.go
!11  Aug 12 13:02:34 247.862156ms/t 0.000214s/u 0.000589s/s ./hello.go
--I-- Aug 12 13:02:41(0.000144653s)
```

社長：CPUを使ってる時間はほぼ同等ですから、あとはバイナリがキャッシュにあるかどうかの違いが支配的ですね。

開発：CPUを一ミリ秒も食ってないのに起動・終了に3ミリ秒かかってしまっているのは、GShellの ForkExecとWaitのせいかも知れません。

雷：ピカドカドッカーングシャバリーン！！！

全員：びっくりしたー！

開発：確認のために、何もしない main()だけのプログラムでもはかってみましたが、同じようなものですね。

```
MacMini% gsh
!1! cat nop.c
int main(int ac,char *av[]){
        return 0;
}
--I-- Aug 12 13:19:40(0.002816583s)
!2! ls -l nop hello-c hello-go
-rwxr-xr-x  1 ysato  staff    12556 Aug 12 13:15 hello-c
-rwxr-xr-x  1 ysato  staff  2182504 Aug 12 13:15 hello-go
-rwxr-xr-x  1 ysato  staff     4248 Aug 12 13:15 nop
--I-- Aug 12 13:19:56(0.005723598s)
!3! ./nop
--I-- Aug 12 13:20:09(0.276314804s)
!4! ./nop
--I-- Aug 12 13:20:10(0.002708361s)
!5! ./nop
--I-- Aug 12 13:20:11(0.003341210s)
!6! hi -l
!0   Aug 12 13:19:40  2.951847ms/t 0.000207s/u 0.000547s/s cat nop.c
!1   Aug 12 13:19:56  5.752756ms/t 0.000209s/u 0.000527s/s ls -l nop he
!2   Aug 12 13:20:09 276.344106ms/t 0.000196s/u 0.000504s/s ./nop
!3   Aug 12 13:20:10   2.73749ms/t 0.000179s/u 0.000498s/s ./nop
!4   Aug 12 13:20:11  3.371625ms/t 0.000192s/u 0.000533s/s ./nop
--I-- Aug 12 13:20:15(0.000084877s)
```

開発：確認のためにzshにて…

```
MacMini% TIMEFMT='%J   %uU user, %uS system, %P cpu, %uE total'
MacMini% time ./nop
./nop   742us user, 1258us system, 67% cpu, 2944us total
MacMini% time ./hello-c
C-Hello GShell!!
./hello-c   780us user, 1949us system, 0% cpu, 318588us total
MacMini% time ./hello-c
C-Hello GShell!!
./hello-c   774us user, 968us system, 58% cpu, 2993us total
MacMini% time ./hello-go
Go-Hello GShell!
./hello-go   1333us user, 1783us system, 9% cpu, 34436us total
MacMini% time ./hello-go
Go-Hello GShell!
./hello-go   1307us user, 1422us system, 56% cpu, 4802us total
MacMini%
```

社長：3ミリ秒がこのMacMiniでのプロセスのfork+execの限界って感じですね。

基盤：この CPU %って、単にCPU時間を経過時間で割ったものみたいですが、便利だと思います。

開発：そうですね。というか、TIMEFMTは他のCPUと互換にします。

＊　＊　＊

社長：さてみなさん、今日の昼食は昨日買ってきたとうもろこしであります。じゃーん。

開発：なんかイマイチ痩せたかんじですね。

基盤：では、レンジで。80度で5分くらいで良いですかね？

社長：もうちょっとかな？

基盤：ともかくレンジでGo!

レンジ：ゴー・・・

基盤：な、なんかめっちゃいい匂いがして来ました。

開発：ふくらむ期待・・・

レンジ：チーン

基盤：ぱかっと。あちっ、ムキムキ。あちちち。ちー。剥けました。

基盤：どれ一口。むしゃ。…　ややナマっぽい。けど、うまーい！

開発：これはいけますねー。むしゃむしゃ。

社長：すばらしい。コンビニで売ってる茹でて長期保存のとは世界が違う。本物の世界ですね。むしゃむしゃ。

基盤：これは醤油をたらして焼きを加えたら最高でしょうね。

開発：ベランダに七輪を置きましょうかね。

社長：「写真の前に挿した桜の花かげに　すずしく光るレモンを今日も置かう」智恵子抄はこの詩で終わってほしかった。

＊　＊　＊

開発：ふぁ。あ。あー。いつの間にか、よく寝てしまいました。何かやろうと思ってた事を忘れてしまい…

基盤：スイカうまいですね。シャグ。

社長：今度は丸ごとに挑戦しますかね。

開発：外れたら悲しいですが、そこがワクワクもしますよね。

＊　＊　＊

開発：それで、今朝方やってたパイプ関係の話ですが。プリミティブ的には、pipeを作る。pipeを入力とするプロセスを生成する。pipeを出力とするプロセスを生成する。これだけだと思います。

開発：たとえばこんな風になります。

```
MacMini% gsh
!1! pipe
--I-- pipe()=[#3,#4](<nil>)
--I-- Aug 12 18:23:31(0.000033453s)
!2! -i #3 bg cat -n
--I-- Opened [3] #3
--I-- in Background [17705]
--I-- Aug 12 18:26:31(0.001769660s)
!3! -o #4 date
--I-- Opened [4] #4
     1  Wed Aug 12 18:26:40 JST 2020
--I-- Aug 12 18:26:40(0.003321584s)
!4! -o #4 date
--I-- Opened [4] #4
     2  Wed Aug 12 18:26:41 JST 2020
--I-- Aug 12 18:26:41(0.002258659s)
!5! -o #4 date
--I-- Opened [4] #4
     3  Wed Aug 12 18:26:43 JST 2020
```

社長：まあ、送る側も受け取り側も、途中で別人に交代しても悪くないのは確かですが。

開発：これで、cat a b c は必要なくなって、cat a ; cat b ; cat c でも良くなります。

基盤：猫いらず。

開発：それ次に、tee を考えます。

社長：cat は fan-in ですが tee は fan-out ですね。

開発：で、こういう風にプリミティブに分解してやると、接続関係は自習になります。メッシュ状にもループ状にも。

社長：私はそれで昔、π言語を使って接続を書きました。

基盤：端末から入力したい気はあまり起こらないですね。

開発：なにか新しい使い方が思いつくかもですよ。たとえば入力した数値を一つカウントダウンして出力するというプログラム。decとしましょう。

```
MacMini% cat dec.c
#include <stdio.h>
#include <stdlib.h>
int main(int ac, char *av[]){
        int cnt;
        if( 1 < ac ){
                cnt = atoi(av[1]);
                fprintf(stderr,"out(%d)\n",cnt);
                printf("%d\n",cnt); fflush(stdout);
        }
        while( 1 ){
                scanf("%d",&cnt);
                fprintf(stderr,"in(%d)\n",cnt);
                if( cnt <= 0 ){
                        break;
                }
                cnt--;
                fprintf(stderr,"out(%d)\n",cnt);
                printf("%d\n",cnt); fflush(stdout);
        }
        return 0;
}
```

社長：なかなか本格的なプログラムですねｗ

開発：で、このプログラムの出力を自分の入力につなげてやると。

```
MacMini% gsh
!1! pipe
--I-- pipe()=[#3,#4](<nil>)
--I-- Aug 12 19:20:48(0.000070382s)
!2! < #3 > #4 dec 5
--I-- Opened [3] #3
--I-- Opened [4] #4
out(5)
in(5)
out(4)
in(4)
out(3)
in(3)
out(2)
in(2)
out(1)
in(1)
out(0)
in(0)
--I-- Aug 12 19:21:00(0.258113200s)
```

基盤：-i と -o じゃなくて < と > になったんですね。

開発：まあ、どっちでもいいんですが。脳内に刻まれてしまった特殊記号には勝てないですね。一文字だし。

社長：なんにしても、普通の shell では書けないことのようには思われます。

開発：namedpipe で出来るとは思いますけどね。

社長：socketpair なら双方向に通信できるから、何か面白いことができるかもですね。

開発：まあいずれにしても、標準入力を処理して標準出力に出すというある意味無敵な汎用モデルには勝てないですけどね。ただ、フィードバックループを持つプログラムは普通にありますから、それをshellでサポートしてコマンドレベルで部品にできれば意味があるようにも思います。

社長：標準制御入力とか標準ログ出力とかもあってよいですよね。要は、入力をselectしてどこからデータが来たらどうする、というのがプログラムの標準モデルとして提供されていないのが問題な気がします。

開発：そのへんは当然シミュレーション用の言語とかではあるわけですから、それをそのままshellに持ち込むでも良いかもですね。

社長：私が学生の時の研究室が貧しくて、シミュレーション用の言語の処理系が買えなかったんです。だもんでSimulaのサブセットを自分で付くて使ったりしてました。

基盤：コロナでお亡くなりになってしまった方ですね。

＊　＊　＊

社長：ところで、どうも気に入った壁掛け時計が無いんで、自作したいなと思うんですが。

基盤：気温とか、電力消費とかも表示すると良いですね。

社長：いつものデスクトップに表示するのは、ちょっと違うかなと思うわけです。

開発：なんてったって憧れるのは電光掲示板ですよね。やはりLEDの発光には魅入られます。小さいのなら 3000円位で手に入るみたいですが。こういうの。

開発：ラズパイで制御したいと思います。

社長：64 x 32 ってことは、ちっちゃいアイコンの情報量ですね。

基盤：ちっちゃいLCDモニタなら5000円くらいでありますよね。ああ、HDMIで7インチ3000円近辺がふつうみたいですが。

社長：そういうの、前の職場で使ってましたが、普通に良かったです。LEDにしましょう。

基盤：タッチパネル機能付きで7000円のもありますね。ラズパイ愛好者にオススメ。

開発：その写真の手の人はコビトさんですかね？

社長：これにしましょう。

経理：溜まってるアマゾンポイント全投入します。ぽちっ。金曜日にお届けとのこと。

基盤：楽しみ！

社長：今気づいたんですが、今日は水曜日なんですね。昨日は月曜だと思ってボウリングの試合に出かけたのですが…

開発：どこかで一日タイムスリップしたようです。

基盤：ブログに空白日は無いようですが…

社長：試合を休んでチームに迷惑をかけてしまいました。今たぶん、2位につけているはずなのですが…

＊　＊　＊

社長：ところで、gshell から簡単に unfs3 を起動できると良いですね。listen localhost:9999 repeat accept unfs3 / $HOME/public-nfs/ みたいな感じで。

開発：unfs3 から GShell の内部状態を見れないと面白くないですね。まだ GoでCプログラムと動的リンクをやってみたことはありませんが…

社長：外部プログラムを起動するのに3ミリ秒かかってしまうようですし、やはり頻繁に使ったり高速な応答が必要な「外部コマンド」は動的リンクして実行する必要があると思います。

開発：考えてみればタイムスリッパでGoのgettimeofday()を簡単に入れ替えられたわけですから、あれと同じですよね。exec() をすり替えるとかですかね…明日トライしましょう。とりあえず今日はここまでで。

— 2020-0812 SatoxITS

GShell_0.0.7-by-SatoxITS ダウンロード