

M2PCCI

Gaëtan Lagier

Eric Thierry



CR TP Système Triangle de Pascale

1.1) Compilation

La commande `./triangle 12 4` génère le résultat suivant en console :

```
newpath 24 245 moveto 38 245 lineto 38 260 lineto stroke newpath 24 245 moveto 24 260 lineto 38 260
lineto stroke /Courier findfont 12 scalefont setfont newpath 25 248 moveto (1) show
newpath 24 230 moveto 38 230 lineto 38 245 lineto stroke newpath 24 230 moveto 24 245 lineto 38 245
lineto stroke /Courier findfont 12 scalefont setfont newpath 25 233 moveto (1) show
newpath 38 230 moveto 52 230 lineto 52 245 lineto stroke newpath 38 230 moveto 38 245 lineto 52 245
lineto stroke /Courier findfont 12 scalefont setfont newpath 39 233 moveto (1) show
newpath 24 215 moveto 38 215 lineto 38 230 lineto stroke newpath 24 215 moveto 24 230 lineto 38 230
lineto stroke /Courier findfont 12 scalefont setfont newpath 25 218 moveto (1) show
newpath 38 215 moveto 52 215 lineto 52 230 lineto stroke newpath 38 215 moveto 38 230 lineto 52 230
lineto stroke /Courier findfont 12 scalefont setfont newpath 39 218 moveto (2) show
newpath 52 215 moveto 66 215 lineto 66 230 lineto stroke newpath 52 215 moveto 52 230 lineto 66 230
lineto stroke /Courier findfont 12 scalefont setfont newpath 53 218 moveto (1) show
newpath 24 200 moveto 38 200 lineto 38 215 lineto stroke newpath 24 200 moveto 24 215 lineto 38 215
lineto stroke /Courier findfont 12 scalefont setfont newpath 25 203 moveto (1) show
newpath 38 200 moveto 52 200 lineto 52 215 lineto stroke newpath 38 200 moveto 38 215 lineto 52 215
lineto stroke /Courier findfont 12 scalefont setfont newpath 39 203 moveto (3) show
newpath 52 200 moveto 66 200 lineto 66 215 lineto stroke newpath 52 200 moveto 52 215 lineto 66 215
lineto stroke /Courier findfont 12 scalefont setfont newpath 53 203 moveto (3) show
newpath 66 200 moveto 80 200 lineto 80 215 lineto stroke newpath 66 200 moveto 66 215 lineto 80 215
lineto stroke /Courier findfont 12 scalefont setfont newpath 67 203 moveto (1) show
```

1.2) Redirection de la sortie dans un fichier :

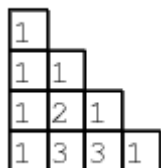
L'utilisation du « > » permet de rediriger la sortie standard, dans notre cas la redirection se fait dans un fichier :

```
./triangle 12 4 > triangle_pascal_12_4.ps
```

Pour visualiser le fichier généré par la commande précédente nous utilisons le logiciel gv :

```
gv triangle_pascal_12_4.ps
```

On obtient le résultat suivant :



1			
1	1		
1	2	1	
1	3	3	1

1.3) Redirection de l'entrée standard :

L'utilisation du « < » permet de rediriger l'entrée standard. Par défaut la commande `tr` lit des caractères tapés au clavier. La commande suivante utilise le fichier `Makefile` comme entrée.

```
tr "bc" "BC" < Makefile
```

Ci-dessous les trois premières lignes du `Makefile` après exécution de la commande :

```
# CeCi est un Commentaire
# Les espaCes en deBut de ligne avant les Commandes de generation
# sont des taBulations, pas des espaCes ordinaires
...
```

1.4) Tube et chainage de commandes :

La commande suivante : `./triangle 11 5 | gv -` permet de rediriger directement la sortie standard de notre programme triangle vers l'entrée standard du logiciel gv. Le « - » après gv indique au programme de lire sur l'entrée standard.

On obtient directement dans gv le résultat suivant :

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	

2) Lancement de l'interprète Postscript par execve() :

Ci-après le fichier affiche_triangle.c modifié :

```
86 /*****
87 /* Le main qui appelle tout le reste ... */
88 /*****
89
90 int main(int argc, char *argv[], char *envp[])
91 {
92
93     char prog_suivant [] = "/usr/bin/gv";
94     char *arguments [] = {"gv", "triangle.ps", NULL};
95
96     unsigned int taille_police, nb_lignes;
97     char nom_executable[200];
98
99     lire_args(argc, argv, 3, message_usage,
100             "%s", nom_executable, "",
101             "%d", &taille_police, "taille de police incorrecte",
102             "%d", &nb_lignes, "nombre de lignes incorrect");
103
104     /* Ici il faudrait ajouter une verification des valeurs */
105     /* de taille_police [8,24] et nb_lignes [1,MAX_LIGNES] */
106
107     sortie = "stdout";
108     taille_triangle = nb_lignes;
109     postscript_triangle (taille_police);
110     //sleep (3);
111
112     execve (prog_suivant, arguments, envp);
113
114     /* On ne doit jamais arriver ici si execve reussit */
115     #ifdef PRINTERERROR
116     printf ("%s\n", strerror (errno));
117     #endif
118     fprintf (stderr, "Je n'ai pas réussi a lancer l'execution du fichier %s", prog_suivant);
119     return 0;
120 }
```

1 : insertion des variables nécessaire à la commande execve()

prog_suivant[] est le chemin d'accès de la fonction gv.

*arguments[] est un tableau de 2 chaînes de caractères + la marque de fin (NULL), la première chaîne de caractère est le nom sous lequel on appelle le fichier gv. La deuxième chaîne de caractères est le premier paramètre passé à la commande gv.

2 : ajout de la fonction execve executant gv, ainsi que la gestion d'erreur de cette fonction.

3) Lancement de gv par fork et exec

Ci-après le fichier affiche_triangle.c modifié :

```
88 /*****
89  */
90 /*****
91
92 int main(int argc, char *argv[], char *envp[])
93 {
94
95     char prog_suivant [] = "/usr/bin/gv";
96     char *arguments [] = {"gv", "triangle.ps", NULL};
97     unsigned int taille_police, nb_lignes;
98     char nom_executable[200];
99
100     pid_t p;
101     int status;
102
103     lire_args(argc, argv, 3, message_usage,
104             "%s", nom_executable, "",
105             "%d", &taille_police, "taille de police incorrecte",
106             "%d", &nb_lignes, "nombre de lignes incorrect");
107
108     /* Ici il faudrait ajouter une verification des valeurs */
109     /* de taille_police [8,24] et nb_lignes [1,MAX_LIGNES] */
110
111     sortie = "stdout";
112     taille_triangle = nb_lignes;
113     postscript_triangle (taille_police);
114     //sleep (3);
115
116     p = fork();
117     if (p < 0){
118         fprintf(stderr, "fils non créé");
119         exit(1);
120     }
121     if (p == 0){
122         execve (prog_suivant, arguments, envp);
123         /* On ne doit jamais arriver ici si execve reussit */
124         #ifdef PRINTERROR
125         printf ("%s\n", strerror (errno));
126         #endif
127         fprintf (stderr, "Je n'ai pas reussi a lancer l'execution du fichier %s", prog_suivant);
128         exit(1);
129     } else {
130         wait(&status);
131     }
132     if(status == 0){
133         fprintf(stderr, "Generation et affichage du triangle de Pascal termine\n");
134     }else{
135         fprintf(stderr, "Generation et affichage du triangle de Pascal impossible\n");
136     }
137
138     return 0;
139 }
```

1 : les variables nécessaires aux fonctions fork() et wait().

2 : création du fils par l'appel à la fonction fork(). Le fils avec le pid = 0 exécute la fonction excve() ajoutée lors de la question précédente. Le père attend la terminaison du fils par l'intermédiaire de la fonction wait(). Un message indique si le fils s'est terminé correctement.

Si p est < 0, le fils n'a pas été créé et envoie d'un message d'erreur.

Les messages que l'on affiche en console passent par la sortie standard d'erreur (stderr) car la sortie standard est redirigée vers le fichier Postscript.

4) Génération du Postscript dans un fichier

La taille du fichier généré par la commande : `./triangle 10 8 > triangle.ps`
-rw-r--r-- 1 thierrye ima-all **6781** févr. 2 19:02 triangle.ps

Après avoir remplacé l'affectation de sortie dans la procédure main ("stdout" -> "triangle.ps"), on observe un seul carré dans la fenêtre gv

1

Et la taille du fichier triangle.ps est beaucoup plus petite :

-rw-r--r-- 1 thierrye ima-all **193** févr. 2 19:18 triangle.ps

L'affichage éroné du triangle de Pascale est due au mode d'ouverture du fichier dans `boite_chiffre.c`. En effet ce dernier est ouvert en mode ré-écriture (w) pour chaque ajout d'un carré. Son contenu est donc effacé à chaque ouverture. L'unique carré visualisé dans gv est le dernier carré calculé et ajouté au fichier.

Pour palier ce problème, il est nécessaire de passer le mode d'ouverture du fichier en mode ajout avec écriture à la fin du fichier (a) :

```
63 if (strcmp (sortie, "stdout") == 0)
64     fsortie = stdout;
65 else {
66     fsortie = fopen (sortie, "a");
67     if (fsortie == NULL)
68         erreur ("Impossible d'ouvrir le fichier en ecriture\n");
69 }
```

A la suite de cette modification on obtient le triangle suivant :

1									
1	1								
1	2	1							
1	3	3	1						
1	4	6	4	1					
1	5	10	10	5	1				
1	6	15	20	15	6	1			
1	7	21	35	35	21	7	1		

Un nouveau problème apparaît, on peut l'observer en modifiant la taille de la police et en inspectant la taille du fichier triangle.ps :

-rw-r--r-- 1 thierrye ima-all **13441** févr. 2 19:35 triangle.ps

	1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
--	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

On constate que la taille du fichier augmente à chaque execution de la commande triangle, et qu'il y a superposition de triangles (voir figure précédente). Pour corriger ce problème il est nécessaire de vider le fichier triangle.ps à chaque lancement du programme triangle dans la fonction main.

```
103 FILE *fsortie; 1
104
105 lire_args(argc,argv,3,message_usage,
106         "%s",nom_executable,"",
107         "%d",&taille_police,"taille de police incorrecte",
108         "%d",&nb_lignes,"nombre de lignes incorrect");
109
110 /* Ici il faudrait ajouter une verification des valeurs */
111 /* de taille_police [8,24] et nb_lignes [1,MAX_LIGNES] */
112
113 sortie = "triangle.ps";
114 fsortie = fopen (sortie, "w");
115 if (fsortie == NULL){
116     fprintf (stderr, "Impossible d'ouvrir le fichier en ecriture\n");
117     exit(1);
118 }
119 fclose (fsortie); 2
```

Solution proposée :

1 : variable nécessaire pour récupérer le retour de la fonction fopen().

2 : Ouverture du fichier en mode ré-écriture (w), s'assurer du bon déroulement de l'ouverture et fermeture du fichier.

On affiche alors un seul triangle et la taille du fichier reste constante après chaque exécution :

```
-rw-r--r-- 1 thierry ima-all 6781 févr.  2 19:53 triangle.ps
```

5) Création d'un tube entre triangle et gv :

Voir image page suivante contenant les modifications apportées à la fonction main du fichier affiche_triangle.c afin de réaliser le tube :

1 : variables créées pour l'utilisation des deux fonctions fork() (pid_t pDroit, pGauche), des deux fonctions wait() (int statusGauche, statusDroit) et de la fonction pipe (int t[2]).

2 : la fonction pipe renvoie dans t deux indices de la table locale des fichier ouvert. Ceux ci correspondent aux pseudo-fichiers représentant l'entrée et la sortie du tube.

Après le premier appel de la fonction fork(), le fils Gauche ferme dans sa table la sortie du tube (close(t[0])). Il redirige ensuite sa sortie standard sur l'entrée du tube (dup2(t[1],1)). Puis réalise l'appel à postscript_triangle(). Le fils gauche est ensuite terminé par l'appel à la fonction exit().

Après le deuxième appel de la fonction fork(), le fils Droit ferme dans sa table l'entrée du tube (close(t[1])). Il redirige ensuite son entrée standard sur la sortie du tube (dup2(t[0],0)). Puis fait appel à execve() pour executer le programme gv.

Le père ferme l'entrée et la sortie du tube et attend la fin des deux processus fils créés. Dans le cas où au moins un des fils ne s'est pas terminé correctement, un message d'erreur est affiché.

```

94 int main(int argc, char *argv[], char *envp[])
95 {
96
97     char prog_suivant [] = "/usr/bin/gv";
98     char *arguments [] = {"gv", "-", NULL};
99     unsigned int taille_police, nb_lignes;
100     char nom_executable[200];
101
102     pid_t pDroit, pGauche;
103     int statusGauche, statusDroit; 1
104
105     int t[2];
106
107     lire_args(argc, argv, 3, message_usage,
108         "%s", nom_executable, "",
109         "%d", &taille_police, "taille de police incorrecte",
110         "%d", &nb_lignes, "nombre de lignes incorrect"); 2
111
112     pipe(t);
113
114     pGauche = fork();
115     if (pGauche < 0){
116         fprintf(stderr, "fils non créé");
117         exit(1);
118     }
119     if (pGauche == 0){
120         close(t[0]); // fermeture de la sortie du tube
121         dup2(t[1], 1); // association de l'entrée du tube avec la sortie standard
122         sortie = "stdout";
123         taille_triangle = nb_lignes;
124         postscript_triangle (taille_police);
125         exit(0);
126     }
127
128     pDroit = fork();
129     if (pDroit < 0){
130         fprintf(stderr, "fils non créé");
131         exit(1);
132     }
133     if (pDroit == 0){
134         close(t[1]); // fermeture de l'entrée du tube
135         dup2(t[0], 0); // association de la sortie du tube avec l'entrée standard
136         execve (prog_suivant, arguments, envp);
137         /* On ne doit jamais arriver ici si execve réussit */
138         #ifdef PRINTERROR
139         printf ("%s\n", strerror (errno));
140         #endif
141         fprintf (stderr, "Je n'ai pas réussi a lancer l'execution du fichier %s", prog_suivant);
142         exit(1);
143     }
144     close(t[0]);
145     close(t[1]);
146     waitpid(pGauche, &statusGauche, 0);
147     waitpid(pDroit, &statusDroit, 0);
148
149     if(statusDroit == 0 && statusGauche == 0){
150         fprintf(stderr, "Generation et affichage du triangle de Pascal termine\n");
151     }else{
152         fprintf(stderr, "Generation et affichage du triangle de Pascal impossible\n");
153     }
154     return 0;

```

6) Utilisation d'un processus par boîte :

Ci-dessous les modifications apportées à la fonction `trace_boite_chiffre()` afin de générer un processus par création de boîte :

```
34
35 /*****
36 /* Pour tracer les appels de boite_chiffre
37 /*****
38
39 void trace_boite_chiffre (unsigned int taille_police, char *sortie,
40                          unsigned int x, unsigned int y,
41                          unsigned int delta_x, unsigned int delta_y,
42                          unsigned int valeur, unsigned int nb_car)
43 {
44     pid_t p; 1
45
46     sprintf (str_taille_police,"%d",taille_police);
47     liste[2] = sortie;
48     sprintf (str_x,"%d",x); sprintf (str_y,"%d",y);
49     sprintf (str_delta_y,"%d",delta_y); sprintf (str_delta_x,"%d",delta_x);
50     sprintf (str_valeur,"%d",valeur); sprintf (str_nb_car,"%d",nb_car);
51
52     p = fork();
53
54     if (p < 0){
55         fprintf(stderr, "bug");
56         exit(1);
57     }
58     if (p == 0){
59         execve(liste[0], liste, NULL);
60         /* On ne doit jamais arriver ici si execve reussit */
61         #ifdef PRINTERROR
62         printf ("%s\n",strerror (errno));
63         #endif
64         fprintf (stderr, "Je n'ai pas reussi a lancer l'execution du fichier %s",liste[0]);
65     }
66
67     close(tGlobal);
68     wait(NULL);
69 }
70 }
```

Nous avons choisi de modifier la fonction `trace_boite_chiffre()` car c'est dans celle ci que l'appel à la fonction `creer_boite()` est réalisé. Dans notre cas, nous changeons cet appel par un appel à la fonction `execve()` qui execute le fichier executable `./creer_boite`. De plus, l'entrée du tube est fermée dans le processus père. Pour ce faire, nous utilisons la variable globale `tGlobal` contenant l'id de l'entrée du tube.

1 : Ajout d'une variable de type `pid_t` pour l'appel à la fonction `fork()`.

2 : Appel à la fonction `fork()` et traitement du cas d'erreur. Le fils créé exécute le fichier executable « `creer_boite` ». Les arguments de la fonction `execve()` sont :

- `liste[0]` = « `./creer_boite` »
- `liste` = tableau de caractères contenant le nom de l'executable et les différents paramètres nécessaires à son exécution.
- `NULL` (en théorie `envp`).

Nous vérifions que chaque appel au fichier executable `creer_boite` créé un nouveau processus avec la commande : `./triangle 12 3`

```
Processus 7495 (pere 7493) : ./creer_boite 12 stdout 24 230 14 15 1 1
Processus 7496 (pere 7493) : ./creer_boite 12 stdout 24 215 14 15 1 1
Processus 7497 (pere 7493) : ./creer_boite 12 stdout 38 215 14 15 1 1
Processus 7498 (pere 7493) : ./creer_boite 12 stdout 24 200 14 15 1 1
Processus 7499 (pere 7493) : ./creer_boite 12 stdout 38 200 14 15 2 1
Processus 7500 (pere 7493) : ./creer_boite 12 stdout 52 200 14 15 1 1
Generation et affichage du triangle de Pascal termine
```