Gestion des processus Linux

Table des matières

RECEPTEUR

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void gestionnaire (int sig);
int main( int argc, char **argv)
{
	int k = 0;

	if(signal (SIGUSR1, gestionnaire) == SIG_ERR)
	{
		fprintf(stderr, "erreur d'association\n");
		exit(0);
	}
	if(signal (SIGUSR2, gestionnaire) == SIG_ERR)
	{
		fprintf(stderr, "erreur d'association\n");
		exit(0);
	}
	printf("processus cible %d\n",getpid());

	while(1)
	{
		k++;
		printf("entree en pause : %d\n", k);
		system("date +\"%H:%M:%S:%N\"");
		pause();
		printf("sortie de pause : %d\n", k);
		system("date +\"%H:%M:%S:%N\"");
	}

	return EXIT_SUCCESS;
}
```

```
void gestionnaire (int sig)
{
	if(sig == SIGUSR1)
	{
		printf("signal SIGUSR1 recu\n");
		system("date +\"%H:%M:%S:%N\"");
	}
	else if (sig == SIGUSR2)
	{
		printf("signal SIGUSR2 recu\n");
		system("date +\"%H:%M:%S:%N\"");
	}
}
```

BOUCLE INFINIE

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char **argv)
{
	while(1)
	{}
	printf("Bonjour\n");

	return EXIT_SUCCESS;
}
```

BOUCLE INFINIE COEUR

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>

int main (int argc, char **argv)
{
	cpu_set_t cpuset;
	int cpu;

	if (argc !=2)
	{
		printf("taper : ./BoucleInfCoeur num \n");
		exit(0);
	}
	cpu = atoi(*++argv);
	printf("a mettre sur coeur : %d\n", cpu);

	printf("Nombre CPU : %d\n",
		sysconf(_SC_NPROCESSORS_ONLN));
```

```
	printf("pid : %d : CPU Affecte : %d\n",
		getpid(), sched_getcpu());

	CPU_ZERO(&cpuset);
	CPU_SET(cpu, &cpuset);
	if (sched_setaffinity(0, sizeof(cpu_set_t), &cpuset) !=0)
	{
		printf("erreur de placement\n");
		exit(0);
	}
	printf("pid : %d : CPU final affecte : %d\n", getpid(), sched_getcpu());

	return EXIT_SUCCESS;
}
```

AFFICHAGE BONJOUR

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char **argv)
{
	printf("Bonjour\n");
	return EXIT_SUCCESS;
}
```

EMETTEUR

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int main (int argc, char **argv)
{
	int pid;
	int k = 0;

	if(argc !=2)
	{
		printf("USAGE : emetteur pid\n");
		exit(0);
	}

	pid = atoi(*++argv);
	printf("processus cible %d\n", pid);

	while(1)
	{
		k++;
		sleep(2);
		printf("envoi signal SIGUSR1 : %d \n", k);
		kill(pid, SIGUSR1);
		k++;
		sleep(2);
		printf("envoi signal SIGUSR2 : %d \n", k);
		kill(pid, SIGUSR2);
	}

	return EXIT_SUCCESS;
}
```

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>

int main (int argc, char **argv)
{
	int ret;
	printf("pere avant creation : %d\n", getpid());

	ret = fork();
	if (ret<0)
	{
		printf("erreur de cration\n");
		exit(0);
	}
	else if (ret>0)
	{
		printf("pere : %d de pere %d : retour %d \n", getpid(), getppid(), ret);
		while(1);
		exit(0);
	}
	else
	{
		printf("fils : %d de pere %d : retour %d \n", getpid(), getppid(), ret);
		exit(0);
	}

	return EXIT_SUCCESS;

}
```

ENVOI SIGNAL ENTRE PERE ET FILS

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void gestionnaire (int sig);

int main (int argc, char **argv)
{
	pid_t ret;
	int k = 0;

	printf("A : pere %d : de pere %d \n", getpid(), getppid());
	ret = fork();

	if (ret < 0)
	{
		printf("erreur de creation\n");
		exit(0);
	}

	else if (ret > 0)
	{
		printf("B : pere %d : de pere %d \n", getpid(), getppid());
		while(1)
		{
			sleep(5);
			k++;
			printf("envoi signal SIGUSR1 : %d \n", k);
			kill(ret, SIGUSR1);
		}
	}

	printf("C : fils %d : de pere %d \n", getpid(), getppid());
```

```
	if(signal (SIGUSR1, gestionnaire) == SIG_ERR)
	{
		fprintf(stderr, "erreur d'association\n");
		exit(0);
	}

	while(1)
	{
		k++;
		printf("entree en pause : %d\n", k);
		system("date +\"%H:%M:%S:%N\"");
		pause();
		printf("sortie de pause : %d\n", k);
		system("date +\"%H:%M:%S:%N\"");
	}

	return EXIT_SUCCESS;
}

void gestionnaire (int sig)
{
	if(sig == SIGUSR1)
	{
		printf("signal SIGUSR1 recu\n");
		system("date +\"%H:%M:%S:%N\"");
	}
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (void)
{
	pid_t ret[2];

	printf("A : pere %d :  %d \n", getpid(), getppid()); // getpid() pour id actuel, getppid() pour id du pere

	ret[0] = fork(); // creation fils

	if (ret[0] < 0)
	{
		printf("erreur de creation\n");
		exit(0);
	}

	else if ( ret[0] > 0) //pere
	{
		printf("B : pere %d :  %d \n", getpid(), ret[0]); // ret pour id du fils
		while(1);
		exit(0);
	}

	//code fils
	printf("C : fils %d :  %d \n", getpid(), getppid());

	ret[1] = fork();// creation petitfils

	if (ret[1] < 0)
	{
		printf("erreur de creation\n");
		exit(0);
```

```
        }

        else if ( ret[1] > 0)//code fils
        {
                printf("D : fils %d :  %d \n", getpid(), getppid());
                while(1);
                exit(0);
        }

        //code petitfils
        printf("E : petitfils %d : de pere %d \n", getpid(), getppid());
        while(1);

        return EXIT_SUCCESS;
}
```