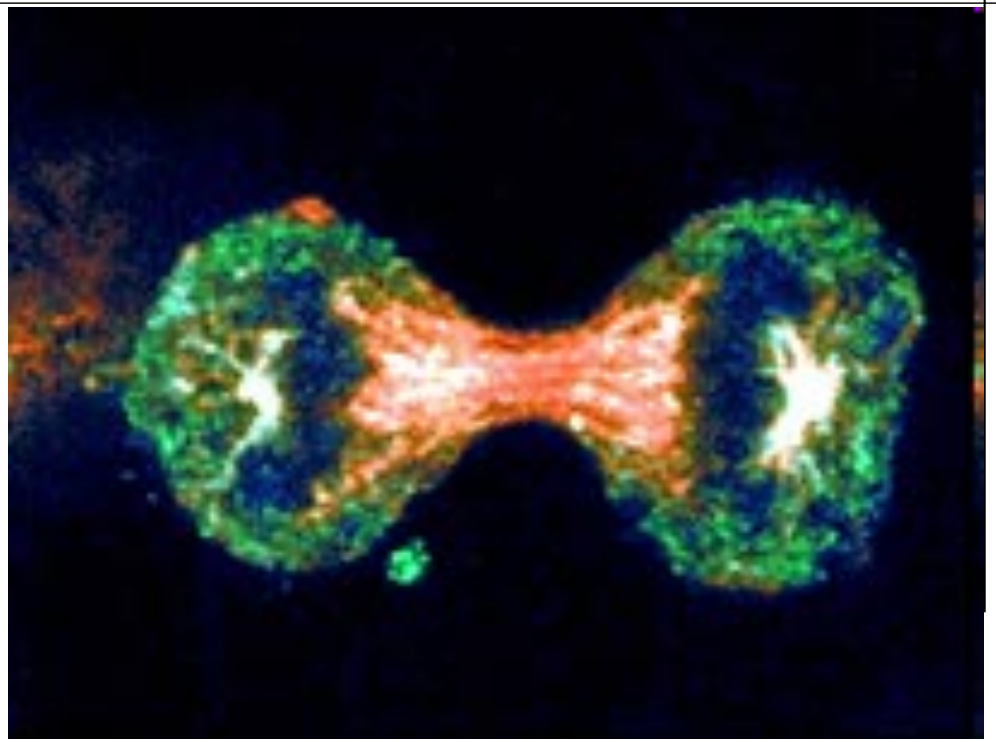
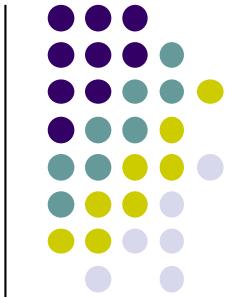
Criação de Processos

Fork( )
Exec( )



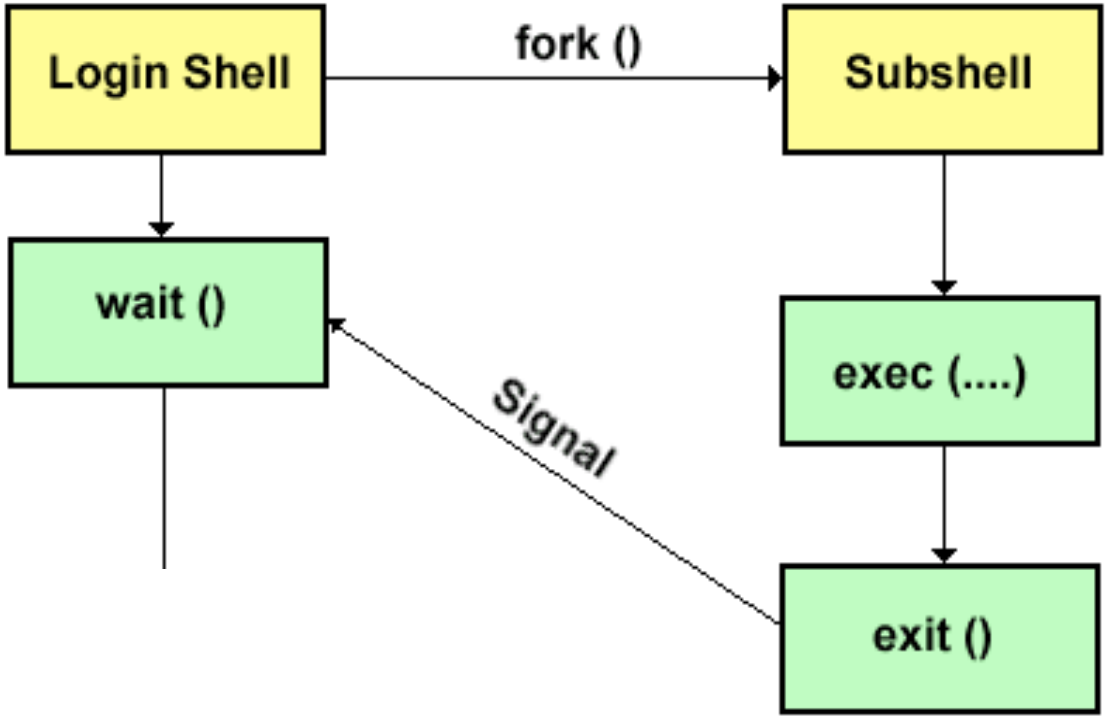
Chamadas de Sistema: Process Management



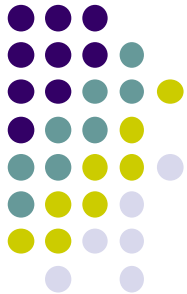
Gerenciamento de processos

Chamada	Descrição
pid = fork( )	Crie um processo filho idêntico ao processo pai
pid = waitpid(pid, &statloc, options)	Aguarde um processo filho terminar
s = execve(name, argv, environp)	Substitua o espaço de endereçamento do processo
exit(status)	Termine a execução do processo e retorne o estado

Chamada fork( ) / exec( )



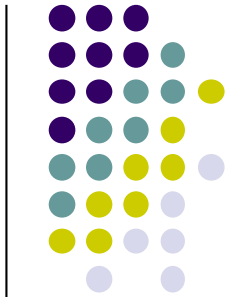
Esboço de uma criação de processo



```
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int mypid, pid, status;
    pid = fork();
    if (pid!=0) { //Pai
        …
       waitpid(-1, &status, 0);
       …
       }
     else { //Filho
        …
       exit(3);
      }
return 0;
}
```

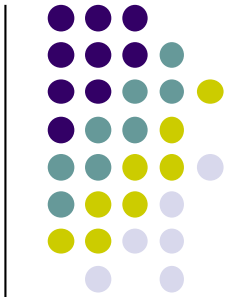
Esboço de uma shell



```
while (TRUE) {                                  /* repeat forever */
    type_prompt( );                             /* display prompt */
    read_command (command, parameters)          /* input from terminal */

if (fork() != 0) {                              /* fork off child process
     */
    /* Parent code */
    waitpid( -1, &status, 0);                   /* wait for child to exit */
} else {
    /* Child code */
    execve (command, parameters, 0);            /* execute command */
 }
}
```

As variantes de exec()



#include <unistd.h>

int execl(const char **path*, **const char** **arg0*, **... *l**, **(char *)0** **l*);

int execv(const char **path*, **char *const** *argv*[]);

int execle(const char **path*, **const char** **arg0*, **... *l**, (char *)0, char *const *envp*[]**l*);

int execve(const char **path*, **char *const** *argv*[], **char *const** *envp*[]);

int execlp(const char **file*, **const char** **arg0*, **... *l**, **(char *)0** **l*);

int execvp(const char **file*, **char *const** *argv*[]);

Com l: argumentos vêm em uma lista separada por “,’
Com v: vetor de caracteres
Com e: com o vetor de envp exemplo: execle("/usr/bin/monitor", "monitor", NULL, ”HOME=myhome", NULL);
Com p: o executável *file é procurado em todo PATH

Argumentos de linha de comando

Quando um processo filho é criado o processo pai, que fazr o exec, pode passar argumentos para o programa, como se fosse na console (na shell).

>> show-arguments 3 5
argc = 3
argv[0] = “show-arguments”, argv[1] =“3” e argv[2] = “5”

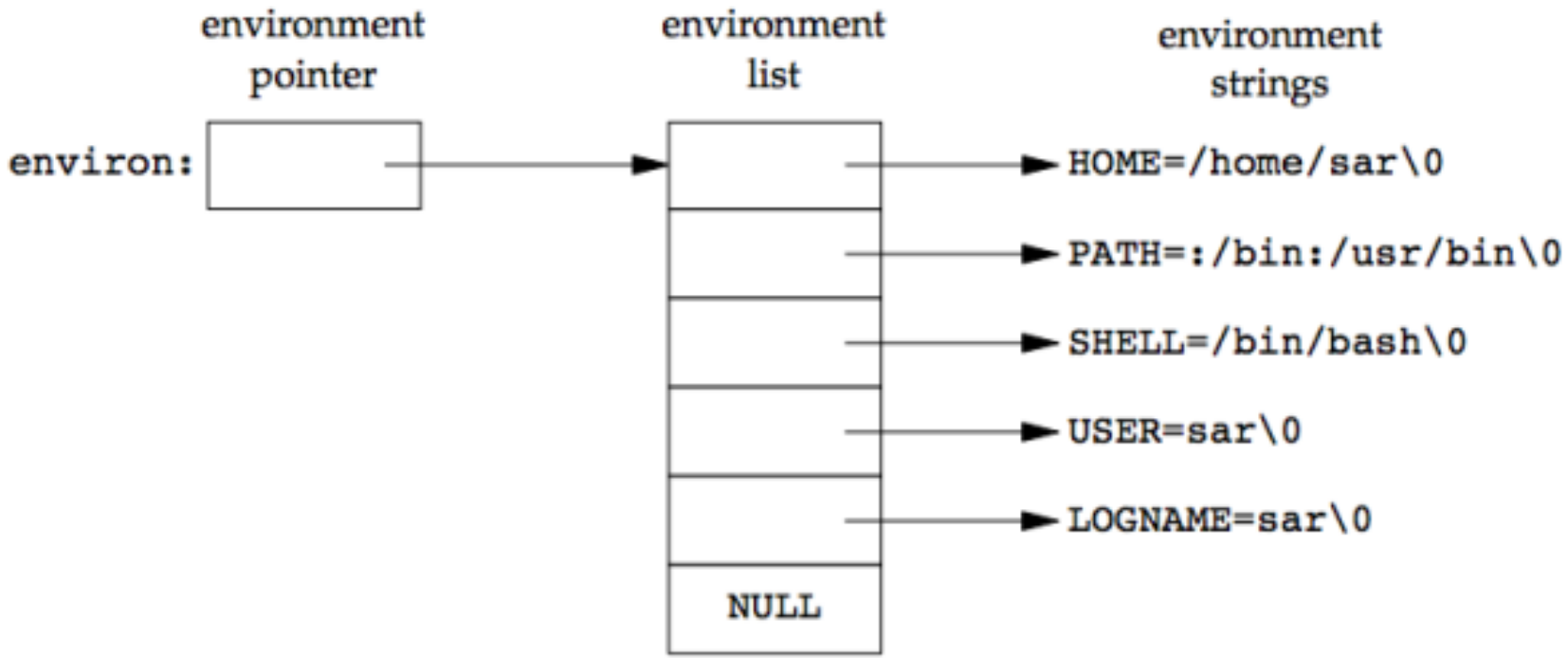
```c
#include "apue.h"

int
main(int argc, char *argv[])
{
    int i;
    for (i = 0; i < argc; i++) /* echo all command-line args */
        printf("argv[%d]: %s\n", i, argv[i]);
    exit(0);
}
```

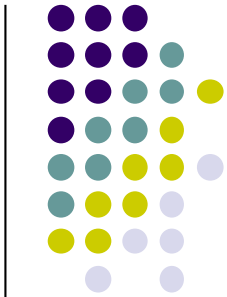
Environment List

- **A cada programa é dado acesso a uma lista de variáveis de ambiente (enrironment variables). No caso de processos criados por fork, inicialmente "herdam" a environment list do processo pai.**

- **Essa lista é acessada via ponteiro environ: e cada elemento da lista é um ponteiro para um endereço de uma string seguindo a convenção VAR-NAME=value\0.**



Relatório Entregável



Um relatório (em um único arquivo, ASCII) c/ sufixo **txt**
contendo, para cada exercício resolvido:

- O número e enunciado do exercício
- Código fonte do programa(s)
- Linha de commando p/ compilação e execução do programa
- A saída gerada
- Um texto com uma reflexão sobre a razão de ter obtido esse resultado

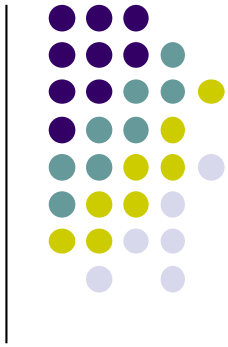
➔ Enviar por email o relatório por email com o título:

[INF1316] Lab # - NomeAluno1, NomeAluno2,

para o monitor e o professor até a meia noite do dia_Lab +1

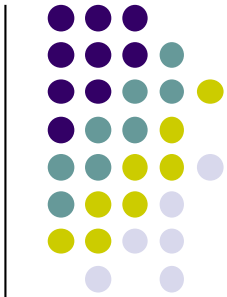
Perguntas?

URL dos Labs



www.inf.puc-rio.br/~endler/courses/inf1019/transp/aulas-praticas

Exercícios



1) Faça um programa para criar dois processos, o pai escreve seu pid e espera o filho terminar e o filho escreve o seu pid e termina.

2) Agora, usando a mesma estrutura de processos pai e filho, declare uma variável visível ao pai e ao filho, no pai inicialize a variável com1 e imprima seu valor antes do fork(). No filho, altere o valor da variável para 5 e imprima o seu valor antes do exit(). Agora, no pai, imprima novamente o valor da variável após o filho ter alterado a variável - após a waitpid(). Justifique os resultados obtidos.

3) Use o programa anterior para ler e ordenar um vetor de 10 posições. O filho ordena o vetor e o pai exibe os dados do vetor antes do fork() e depois do waitpid(). Eles usarão o mesmo vetor na memória? Justifique.

4) Modifique o programa anterior para que o filho execute um programa elaborado por você, que mande imprimir uma mensagem qualquer no vídeo, por exemplo, “alo mundo”. Em seguida altere o programa do item 4 para o filho executar o programa **echo** da shell.