UNIVERSITÉ LUMIÈRE LYON 2
UNIVERSITÉ DE LYON

Département d'Informatique et de Statistique
DIS
Faculté de Sciences Économiques et de Gestion

Système d'exploitation UNIX Éléments de langage C

L3 Informatique Décisionnelle et Statistique
Année 2011-2012
Jérôme Darmont

http://eric.univ-lyon2.fr/~jdarmont/

Plan du cours

- Systèmes d'exploitation
- UNIX
- Langage C

Composition d'un ordinateur

- Un ou plusieurs processeurs universels
- Plusieurs processeurs spécialisés (contrôleurs). Ex. Horloge, contrôleur de disque, carte vidéo…
- Mémoire centrale
- Périphériques (entrées/sorties, stockage)

➲ Nécessité de piloter cet ensemble

Qu'est-ce qu'un système d'exploitation ?

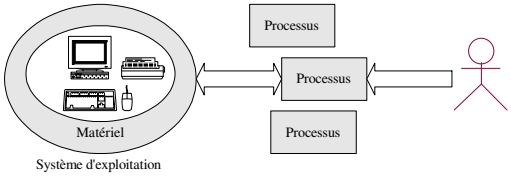
- Définition : Couche logicielle qui permet et coordonne l'utilisation du matériel entre les différents programmes d'application
- Ensemble de procédures qui pilotent le matériel de manière transparente et facile d'utilisation à partir d'un programme (machine virtuelle)

Historique

- Années 40 : Langage machine, pas de système d'exploitation (SE)
- Années 50 : Assembleur, 1ers langages évolués (Fortran, Cobol), traitement par lots
- Années 60 : Multi-programmation, accès interactif, études théoriques sur les SE
- Années 70 : Améliorations des SE (performances, sécurité, convivialité), intégration des réseaux
- Années 80 : Même SE pour des matériels différents, convivialité sans cesse accrue
- Années 90 : Objets distribués (CORBA, DCOM)
- Années 00 : Intégration du Web dans les SE
- Années 10 : Informatique dans les nuages, SE Web

Structure des SE actuels

Système informatique : ensemble de matériel éventuellement réparti sur plusieurs sites + SE



Processus : Activité résultant de l'exécution d'un ou plusieurs programmes en vue de l'exécution d'une tâche bien définie.

Fonctions d'un SE

- Gérer les processus et leur existence simultanée
- Fournir aux processus les moyens d'accès à toutes les ressources dont ils ont besoin
- Offrir une interface de programmation
- Offrir une interface opérateur conviviale
- Optimiser l'utilisation des éléments du système informatique

Plan du cours

- ✓ Systèmes d'exploitation
- UNIX
- Langage C

Historique

- Création en 1969 au laboratoire AT&T Bell, USA par Kenneth Thompson et Denis Ritchie (gestion d'un mini-ordinateur)
- Multi-tâches, multi-utilisateurs en 1973
- Deux principales familles : System V (AT&T) et BSD (Université de Berkeley, Californie)
- Environnement graphique X Window System développé au MIT (Boston) en 1981
- Sur PC, apparition de Linux, écrit par Linus Thorvalds, en 1991 (version d'UNIX du DIS)

Composants typiques d'un UNIX

- Noyau (*kernel*) : ensemble de fonctions système assurant la gestion de la mémoire, des E/S et des processus
- Système de gestion de fichiers hiérarchisé
- Interpréteurs de commande (*shells*)
- Utilitaires : compilateur C, éditeurs, logiciels réseau…
- Système de messagerie électronique

Connexion/Déconnexion

- Connexion
  - Saisie du nom d'utilisateur (*login*)
  - Saisie du mot de passe (*password*)
  - ➲ Accès au compte personnel de l'utilisateur
  - Changer son mot de passe : commande passwd
- Déconnexion
  - Commande exit (shell)
  - Menu adéquat sous X

Information et aide

- whoami affiche le nom de l'utilisateur courant
- who affiche les utilisateurs actuellement connectés
- finger [*login_utilisateur*] affiche des informations sur un utilisateur
- date permet de consulter la date et l'heure
- cal [*année*] affiche un calendrier
- man *commande* permet d'obtenir de l'aide sur une commande. Ex. man cal

Système de fichiers

- Fichier : Document géré par le système et contenant des données (suite d'octets)
- Organisation : Arborescence de répertoires (*directories*). Un répertoire peut contenir des fichiers et des sous-répertoires. Racine de l'arborescence (*root directory*) : / .
- Chemin d'accès : Façon d'atteindre un fichier ⇒ Répertoires parents séparés par des / + nom du fichier. Ex. /usr/bin/who

Système de fichiers

- Chemin absolu : À partir de la racine (/).
- Chemin relatif : À partir du répertoire courant
  Caractères spéciaux : . répertoire courant
  .. répertoire parent
  ~ répertoire personnel
  Ex. ../../bin/who ~/mesdocs/monfich
- Noms de fichiers :
  - 256 caractères maxi
  - Caractères interdits : * ? < > /
  - Noms de fichiers réservés : **.** et **..**
  - Fichiers cachés : Commencent par **.**

Système de fichiers

- Caractères génériques :
  - ? remplace un caractère
  - * remplace un nombre quelconque de caractères
  - [ ] permet de spécifier une plage de caractères
    Ex. [IDS] pour I, D ou S ; [A-M] pour tous les caractères entre A et M
- Permissions d'accès : Protection des fichiers

Ensembles :	*Droits :*
Propriétaire	Lecture (r)
Groupe	Écriture (w)
Autres	Exécution (x)

Système de fichiers

- Exemple de permissions d'accès :
  `-rwxr-x--- 1 jdarmont web 0 Aug 28 15:40 toto`
  - Propriétaire : **RWX** (lecture, écriture, exécution)
  - Groupe : **RX** (lecture, exécution)
  - Autres : aucun droit
- Permissions d'accès sur les répertoires : Identiques à celles des fichiers, mais interprétation légèrement différente (exécution : droit d'activation du répertoire)

Système de fichiers

- ls [*rep*] affiche le contenu d'un répertoire
  Options : -l format d'affichage long
  -a affichage des fichiers cachés
  -R contenu des sous-répertoires
- file *fichiers* affiche le type des fichiers
- cp *source dest* copie des fichiers
- mv *source dest* déplace/renomme des fichiers
- rm *fichiers* efface des fichiers
  Options : -i supprime avec confirmation
  -f force la suppression sans confirmation
  -r détruit récursivement les sous-répertoires

Système de fichiers

- find [*rep*] recherche des fichiers
  Options : -name fichier selon le nom
  -type t selon le type
  -user u selon le propriétaire
  -group g selon le groupe
  -perm p selon les permissions d'accès
  -mtime j fichiers modifiés depuis il y a j jours
- pwd affiche le chemin absolu du répertoire courant
- cd [*rep*] activation du répertoire de base ou du répertoire indiqué si le paramètre est spécifié

Système de fichiers

- mkdir *reps* crée des répertoires
- rmdir *reps* détruit des répertoires (vides)
- chmod *mode fichiers* change les permissions d'accès aux fichiers

Classes :	*Actions :*	*Permissions :*
u propriétaire	= attribuer	r lecture
g groupe	+ ajouter	w écriture
o autres	- retirer	x exécution
a tous		

Ex. chmod a = rwx toto
Mode octal : chmod 777 toto (r = 4, w = 2, x = 1, ∅ = 0)

L'éditeur de texte vi

- vi : Éditeur de texte standard sur tout UNIX
  vi *nom_fichier*
- Deux modes de fonctionnement :
  - Mode édition
  - Mode commande
  - Mode édition → Mode commande : Touche *Escape*
- Quitter vi :
  - :q sortie simple
  - :q! sortie sans sauvegarde
  - :wq sortie avec sauvegarde

L'éditeur de texte vi

- Annulation de commande : u
- Édition :
  - i insérer du texte avant le curseur
  - a insérer du texte après le curseur
  - I insérer du texte en début de ligne
  - A insérer du texte en fin de ligne
  - o insérer du texte sous la ligne courante
  - O insérer du texte au-dessus de la ligne courante

L'éditeur de texte vi

- Déplacement :
  - l, → à droite
  - h, ← à gauche
  - j, ↓ en bas
  - k, ↑ en haut
  - 0 début de ligne
  - $ fin de ligne
  - w début mot suivant
  - b début mot précédent
  - G fin de fichier
  - *n*G ligne numéro *n*
  - CTRL+f 1 écran + bas
  - CTRL+b 1 écran + haut
  - CTRL+d ½ écran + bas
  - CTRL+u ½ écran + haut
  - m*c* marque une ligne *c*
  - '*c* saute à la ligne *c*

L'éditeur de texte vi

- Suppressions / remplacements :
  - x efface le caractère situé sous le curseur
  - *n*x efface *n* caractères à partir du curseur
  - r*c* remplace le caractère situé sous le curseur par *c*
- Copier/coller :
  - dd coupe une ligne (sert aussi à effacer une ligne)
  - :n,m d coupe les lignes numéro n à m
  - yy copie une ligne
  - :n,m y copie les lignes numéro n à m
  - P colle avant le curseur p colle après le curseur

L'éditeur de texte vi

- Recherche :
  - /*mot* recherche la 1ère occurrence de la chaîne *mot*
  - / recherche l'occurrence suivante
- Remplacement :
  - :s/*ch1*/*ch2*/ remplace la 1ère occurrence de la chaîne *ch1* par la chaîne *ch2*

Manipulation de fichiers textes

- cat *fic* affiche le contenu du fichier
- more *fic* idem mais page par page
- head *fic* affiche le début du fichier
- tail *fic* affiche la fin du fichier
- grep *motif fichiers* affiche les lignes des fichiers qui respectent un motif (recherche du motif dans des fichiers)
- sort *fic* trie les lignes du fichier

Impression

- lpr *fichier* imprime le contenu du fichier
- lpq affiche l'état de la file d'impression
- lprm *job* supprime un job de la file d'impression (numéro de job fourni par lpq)

Entrées/Sorties

- Entrée standard : Source de données des commandes (normalement le clavier)
- Sortie standard : Destination des résultats de commandes (normalement l'écran)
- Sortie d'erreur standard : Destination des messages d'erreur des commandes (normalement l'écran)

Entrées/Sorties

- Redirections :
  - < redirige l'entrée à partir d'un fichier
    Ex. passwd < mdp.txt
  - > redirige la sortie vers un fichier
    Ex. ls > dir.txt
  - >> concatène la sortie avec un fichier
    Ex. cat mdp.txt >> dir.txt
  - 2> (ou >&*) redirige les erreurs vers un fichier
    Ex. find ~ -name dir.txt 2> erreur.log
  - 2>> (ou >>&*) concatène les erreurs avec un fichier
    Ex. find ~ -name mdp.txt 2>> erreur.log

*avec certains shells

Entrées/Sorties

- Tubes (*pipes*) :
  - | redirige la sortie d'une commande vers une autre
    Ex. ls | more
  - | tee s1 s2 dédouble la sortie d'une commande
    Ex. ls | tee dir1.txt dir2.txt
    ls –al | tee dir.txt | more

Processus

- Exécution d'une commande en arrière-plan : &
  Ex. xpinguin &
- ps affiche des informations sur les processus en cours d'exécution, notamment leur PID (*process identifier*)
  Options : -l format d'affichage long
  -a affiche tous les processus s'exécutant sur le terminal
- top affiche en temps réel des informations sur les processus
- kill [–9] *pid* arrête un processus

Interpréteurs de commande

- Il existe de nombreux interpréteurs de commandes (*shells*) différents. Les plus courants sous Linux : bash (*Bourne again shell*) et tcsh (extension du *C Shell*).
- Changement de shell : chsh
- Script : Suite de commandes dans un fichier texte avec permission d'exécution (x)
- Fichier d'initialisation : Script exécuté au login (fichier .bashrc ou .tcshrc)

Interpréteurs de commande

- Variables d'environnement : Définies au niveau du shell, accessibles à tout programme
  Ex. PATH, chemin des fichiers exécutables
- Afficher toutes les variables : set ou setenv
- Afficher une valeur : echo $*nom_var*
  Ex. echo $PATH
- Affecter une valeur :
  - bash : export *var=val* Ex. export a=coucou
  - tcsh : setenv *var val* Ex. setenv a coucou

Interpréteurs de commande

- Alias de commande :
  - bashrc : alias *nom_alias=commande*
    Ex. alias l='ls –l'
  - tcsh : alias *nom_alias commande*
    Ex. alias l 'ls –l'
- Paramètres dans les scripts : $1 $2 $3…
  Ex. Appel de la commande toto param1 45 3.14
  Dans le script : $1 a la valeur param1
  $2 45
  $3 3.14

Plan du cours

- ✓ Systèmes d'exploitation
- ✓ UNIX
- Langage C

Généralités

- Création du langage C par Dennis Ritchie pour développer UNIX (1969)
- Normalisation en 1983 par l'ANSI (*American National Standard Institute*)
- Caractéristiques du langage C :
  - Efficacité
  - Concision
  - Portabilité

Structure d'un programme C

Exemple :

```
/* Affiche "Coucou !" à l'écran */

#include <stdio.h>

main() {
    printf("Coucou !\n");
}
```

Commentaire

Bibliothèque

Programme principal

Affichage

Compilation

- Programme source : Écrit dans un fichier texte dont l'extension est en général .c
- Compilation : Commande UNIX cc

  Ex. cc coucou.c
  ⇒ programme exécutable nommé a.out

  Ex. cc coucou.c –o coucourun
  ⇒ programme exécutable nommé coucourun

Types de données

- Types de base :
  - char caractère (1 octet)
  - int entier
  - float réel (nombre en virgule flottante)
  - double réel double précision
- Qualificateurs :
  - short diminution de précision (entiers)
  - long augmentation de précision (entiers et doubles)
  - signed type signé (caractères et entiers)
  - unsigned type non signé (caractères et entiers)

Exemples de déclarations

```
short int i;                /* un entier court */
float a, b, c;              /* trois réels */
char z = 'Z';               /* initialisation */
int j = 2, k = 3, l;        /* initialisations */
const float pi = 3.14;      /* constante */
```


Entrées/Sorties (stdio.h)

- Sorties : printf("*format*", *arg1*, *arg2*…);
- Formats :
  - %c caractère %s chaîne
  - %d entier %u entier non signé
  - %f réel %e réel avec exposant
- Exemples :
  printf("PI = %f \n", pi);
  printf("%d + %d = %d \n", j, k, j + k);
- Entrées : scanf("*format*", &*arg1*, &*arg2*…);
  Ex. scanf("%d", &un_entier);

Opérateurs

- Affectation : = Ex. i = 1;
- Opérateurs arithmétiques :
  - Addition + Soustraction -
  - Multiplication * Division /
  - Modulo %
- Opérateurs de comparaison :
  - Supérieur > Supérieur ou égal >=
  - Inférieur < Inférieur ou égal <=
  - Égal == Différent !=

Opérateurs

- Opérateurs logiques :
  - ET : && OU : || NOT : !
- Incrémentation et décrémentation :
  - Incrémentation : ++ Décrémentation : --
  - Ex. i = 0;
    j = ++i; /* Pré-incrémentation : j = 1, i = 1 */
    k = i++; /* Post-incrémentation : k = 1, i = 2 */
- Affectation + opération : *op*=
  - Ex. i += 2; ⇔ i = i + 2;
    i *= j + 1; ⇔ i = i * (j + 1);

Opérateurs

- Opérateur conditionnel : ?
  - Utilisation : *var* = *cond* ? *res_si_vrai* : *res_si_faux*;
  - Ex.
```
i = 1; j = 2;
k = i < j ? i : j;    /* k vaut 1 */
```
- Conversion de type (*cast*) :
  - Utilisation : *var* = (*type*) *expression*;
  - Ex.
```
i = (int) pi;          /* i vaut 3 */
j = (int) 3.14;        /* j vaut 3 aussi */
```


Structures de contrôle

- Test : if *instr* else *instr*
  - Ex.
```
if (i < j) { k = i; }
else { k = j; } /* Facultatif */
/* Accolades facultatives si une seule instruction */
```
- Sélection : switch (*var*) { *cas* }
  - Ex.
```
switch (un_caractere) {
    case 'a': { i = 1;
                break; }
    case 'b': { i = 2;
                break; }
    default: i = 0; /* Facultatif */
}
```


Structures de contrôle

- Boucle *pour* : for (*expr1*; *cond*; *expr2*) *instr*
  - Ex.
```
for (i = 1; i <= 10; i++) {
    printf("%d \n", i); }
```
- Boucle *tant que* : while (*cond*) *instr*
  - Ex.
```
i = 1;
while (i <= 10) { printf("%d \n", i);
                  i++; }
```
- Boucle *répéter* : do *instr* while *cond*
  - Ex.
```
i = 1;
do {    printf("%d \n", i);
        i++;
 } while (i <= 10);
```


Objets structurés

- Tableau unidimensionnel :
  - Déclaration : *type* mon_tableau[*dimension*]
    Ex. int t[10];
  - Ex. d'utilisation : for (i = 0; i < 10; i++) t[i] = i;
    NB : Le 1er élément d'un tableau a pour indice 0.
- Tableau multidimensionnel :
  - Déclaration : *type* tab[*dim1*][*dim2*]…
    Ex. int matrice[4][5];
  - Ex. d'utilisation :
```
for (i = 0; i < 4; i++)
  for (j = 0; j < 5; j++)
    matrice[i][j] = i * j;
```


Objets structurés

- Chaîne de caractères : Tableau de caractères terminé par le caractère \0
  - Déclaration :
```
char msg[10];
char msg[10] = "Coucou !";
char msg[ ] = "Coucou !";
```
- Structure : struct *nom* { *champs* } *vars*;
  - Ex.
```
struct personne {
    char nom[30];
    char prenom[30];
    char age;
} p1, p2, p3;
```
  - Déclaration séparée de variables de type structure :
    struct personne p1, p2, p3;

Objets structurés

  - Ex. d'utilisation :
```
strcpy(p1.nom, "Darmont");
strcpy(p1.prenom, "Jérôme");
p1.age = 35;
```
- Union : union *nom* { *champs* } *vars*;
  - Ex.
```
union mon_union {
    char octet;
    unsigned short int octets4;
} u;
```
  - Déclaration séparée : union mon_union u;
  - Fonctionnement :
```
u.octet = 1;
```
    u.octet vaut 1 et u.octets4 vaut 48897.

Objets structurés

- Énumérations :
  - Ex. enum jour {lundi, mardi, mercredi} j;
  - Déclaration séparée : enum jour j;
  - Ex. d'utilisation :
    for (j = lundi; j < mercredi, j++) …
- Définition de nouveaux types :
  - Ex.
```
typedef struct un_pays {
        char nom[30]; char langue[30];
} pays; /* pays est le nom du type */
```
  - Déclaration d'une variable de ce type : pays monPI;
  - NB : Fonctionne aussi pour unions et énumérations.

Fonctions

- Prototype : *type_retour nom_fonction* (*paramètres*);
  NB : Déclaration du prototype avant le main( )

  Ex. int somme (int a, int b);

  Le type de retour peut être « rien » (procédure) : void
- Code : *type_retour nom_fonction* (*paramètres*) {…}
  NB : Spécification du code après le main( )

  Ex.
```
int somme (int a, int b) {
        int c = a + b;
        return c;
}
```

  return permet de retourner une valeur à la fonction appelante.

Fonctions

- Appel : Dans le main( )
- Exemple complet :

```
#include <stdio.h>
int somme (int a, int b); /* Prototype */
main() {
      printf("%d + %d = %d \n", 1, 2, somme(1, 2));
}                                                  /* Appel */
int somme (int a, int b) { /* Code */
      return a + b;
}
```


Classes de stockage

- Variables globales : Définies hors de toute fonction (y compris main( )) – À ÉVITER !!!
- Variables locales : Définies dans un bloc { }
- Variables statiques : Variables locales permanentes (conservation d'informations)
  Ex. static int i;
- Variables registres : Le compilateur place la variable dans un registre du processeur (optimisation)
  Ex. register int i;
- Variables externes : Définies dans un autre fichier source
  Ex. extern int i;

Le préprocesseur

- Préprocesseur : Traite le fichier source avant la compilation (commandes #)
- Inclusions :
```
#include <nom_bib.h>
#include "nom_fichier.h"
```
- Macros : #define *nom_macro expr*
  Ex.
```
#define TAILLE_MAX 100 /* Constante symbolique */
#define SOMME(a, b) (a) + (b)
#define MIN(a, b) ((a) < (b) ? (a) : (b))
```
- Annuler la définition : #undef *nom_macro*
  Ex. #undef MIN

Le préprocesseur

- Inclusion conditionnelle : Permet de prendre en compte ou nom une partie du code source en fonction d'une condition

```
#if cond
…
#else
…
#endif
```

  Il est également possible de tester si une macro a été définie ou non.

```
#ifdef                ou        #ifndef
#endif                          #endif
```

Pointeurs

- Pointeur : Variable contenant une adresse mémoire
- Déclaration : type *nom_ptr*;
  Ex. int *pi;
- Manipulation :
  Ex.
```
int i = 3;
pi = &i;            /* pi = adresse de i */
printf("%d", *pi);  /* contenu de pi = 3 */
*pi = 4;            /* contenu de pi = 4 */
printf("%d", i);    /* i = 4 aussi */
```


Pointeurs

- Allocation dynamique de mémoire :
  Ex.
```
int *p;
p = (int*) malloc(sizeof(int));
/* p pointe sur un entier */
free(p); /* libération de la mémoire */
```
  Bibliothèque : stdlib.h
- Passage de paramètre par adresse :
  Ex.
```
void newval(int a, int *b) {
        a = 32; *b = 44; }
```
  Appel : newval(va, &vb);
  va (passée par valeur) aura gardé sa valeur
  vb (passée par adresse) aura changé de valeur

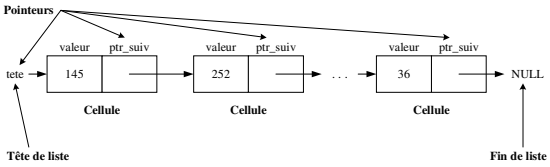
Pointeurs

- Pointeurs et tableaux :
  Ex.
```
int t[10];
int *p;
p = t; ⇔ p = &t[0];   /* t ⇔ &t[0] */
                      /* *(t+2) ⇔ t[2] */
```
- Paramètres de main( ) :
  Ex.
```
#include <stdio.h>
main(int argc, char **argv) {
int i;
for (i = 0; i < argc; i++)
   printf("Argument no %d : %s \n", i, argv[i]);
}
```


Application : Gestion de liste chaînée

Liste chaînée : de taille variable (dynamique)
≠
Liste contiguë (tableau) : de taille fixe

Ex. Liste chaînée d'entiers positifs



Application

```
#include <stdio.h>
#include <stdlib.h> /* Bibliothèque pour la gestion
                       de mémoire dynamique */

/* Définition d'un type cellule */
typedef struct cell {
       int valeur;
       struct cell *ptr_suiv;
} cellule;

/* Programme principal */
main() {
       /* Déclaration de pointeurs */
       cellule *tete; /* Tête de liste */
       cellule *cour; /* Pointeur courant */
       int valsaisie; /* Valeur saisie au clavier */
```


Application

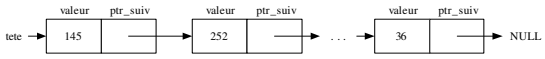
```
/* Création de la liste */

tete = NULL; /* Initialisation à vide */

printf("Saisir une valeur (<0 pour arreter) :");
scanf("%d", &valsaisie);

while (valsaisie >= 0) {
       /* Création nouvelle cellule */
       cour = (cellule*) malloc(sizeof(cellule));
       cour->valeur = valsaisie; /* Stockage valeur */
       cour->ptr_suiv = tete; /* Suivant = tete */
       tete = cour; /* Tete = nouvelle cellule */
       printf("Saisir une valeur (<0 pour arreter) :");
       scanf("%d", &valsaisie);
}
```

Application

```
/* Parcours de la liste */

cour = tete; /* Positionnement sur la tete de liste */
while (cour != NULL) { /* Tant que pas fin de liste */
      printf("%d \n", cour->valeur); /* Affichage val */
      cour = cour->ptr_suiv; /* Cellule suivante */
}

} /* Fin main() */
```

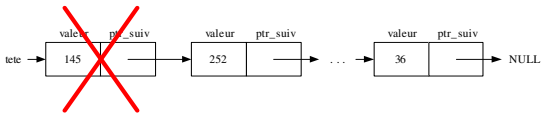

Application (suite) : Destruction de la liste chaînée

Principe



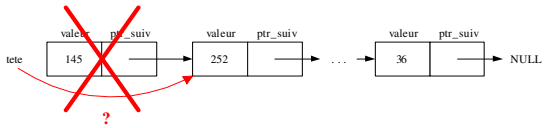
Application (suite) : Destruction de la liste chaînée

Principe



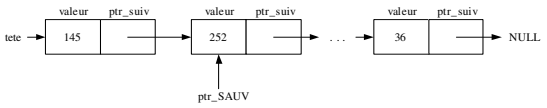
Application (suite) : Destruction de la liste chaînée

Principe



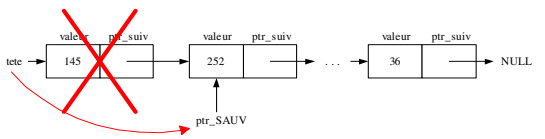
Application (suite) : Destruction de la liste chaînée

Principe



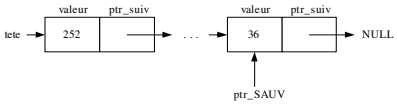
Application (suite) : Destruction de la liste chaînée

Principe



Application (suite) : Destruction de la liste chaînée

Principe



Etc.

Application

```
/* Destruction de la liste */

cour = tete; /* Positionnement sur la tete de liste */
while (cour != NULL) { /* Tant que pas fin de liste */
       sauv = cour->ptr_suiv; /* Sauvegarde du suivant */
       free(cour); /* Destruction cellule courante */
       cour = sauv; /* Cellule suivante */
}
tete = NULL; /* Liste vide */
```


Application

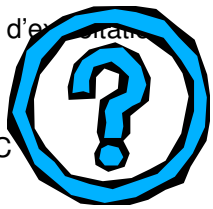
```
/* Destruction de la liste sous forme de fonction */

void destruction_liste(cellule **t) {
       cellule* cour, sauv;
       cour = *t;
       while (cour != NULL) {
              sauv = cour->ptr_suiv;
              free(cour);
              cour = sauv;
       }
       *t = NULL;
}

/* Appel dans le programme principal :
   destruction_liste(&tete);                    */
```


Application

```
/* Destruction de la liste : 2e solution */

cellule* destruction_liste(cellule *t) {
       cellule* cour, sauv;
       cour = t;
       while (cour != NULL) {
              sauv = cour->ptr_suiv;
              free(cour);
              cour = sauv;
       }
       return NULL;
}

/* Appel dans le programme principal :
   tete=destruction_liste(tete);              */
```


Application

```
/* Insertion dans la liste sous forme de fonction */

void insertion_liste(cellule **t, int v) {
       cellule* cour;
       cour = (cellule*) malloc(sizeof(cellule));
       cour->valeur = v;
       cour->ptr_suiv = *t;
       *t = cour;
}

/* Appel dans le programme principal :
   insertion_liste(&tete, valsaisie);          */
```


Plan du cours

- ✓ Systèmes d'exploitation
- ✓ UNIX
- ✓ Langage C