hexagone

École Supérieure d'Informatique

Les signaux

#1 Tâches et terminaux

Processus

- Tâche unitaire, avec contexte autonome
- Capable d’exécuter des actions comme fork, exec, exit, . . .
- Identifié par son pid

Groupe de processus

- Tâche logique regroupant plusieurs processus
- Chaque processus est capable d’exécuter les actions
- Identifié par son pgid
- A au plus un leader (en général le créateur du groupe)
- Quand le leader fait un exit, plus de leader
- Pas de possibilité de recréer un leader

Session

- Ensemble de groupes de processus
  - Contrôlés par une même entité
  - En général un shell associé à un terminal
- Identifié par son sid
- Comprend un ensemble de tâches en avant-plan ou arrière-plan
- A au plus un leader (en général le créateur)

- En mode texte: session = actions du login au logout
- En mode graphique: en général, une session par émulateur de terminal

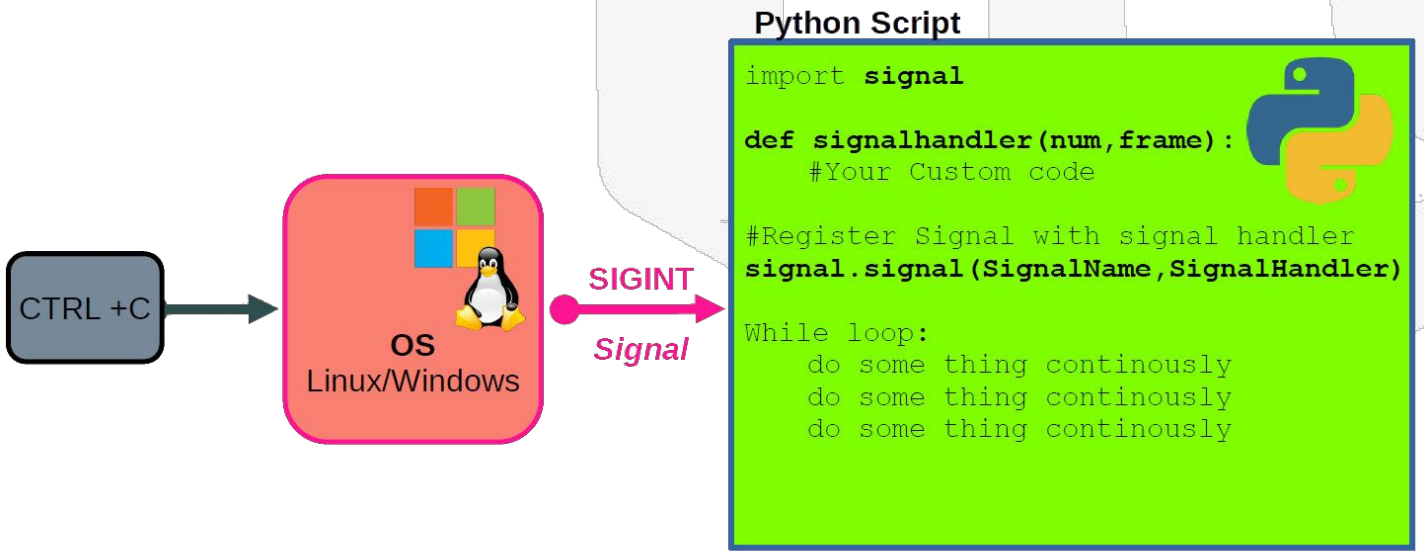
Terminal de contrôle

- Console
  - Virtuelle (/dev/tty*)
  - Port série (/dev/ttyS*)
  - Pseudo terminaux (/dev/pts/*)
- Servent aux entrées-sorties
- Permet la gestion de tâches
- Supportent une session (max un terminal par session)
- Ouverture par session leader sans terminal
- Max un groupe de processus en avant-plan

#2 Signaux

Signaux

- Mécanisme asynchrone de communication
- Permet de notifier un événement
- Ressemble à un mécanisme d'interruption (CPU)
- Un signal est identifié par un numéro
- Définis dans <signal.h> en C et le module Python *signal*



Utilisation d’un signal

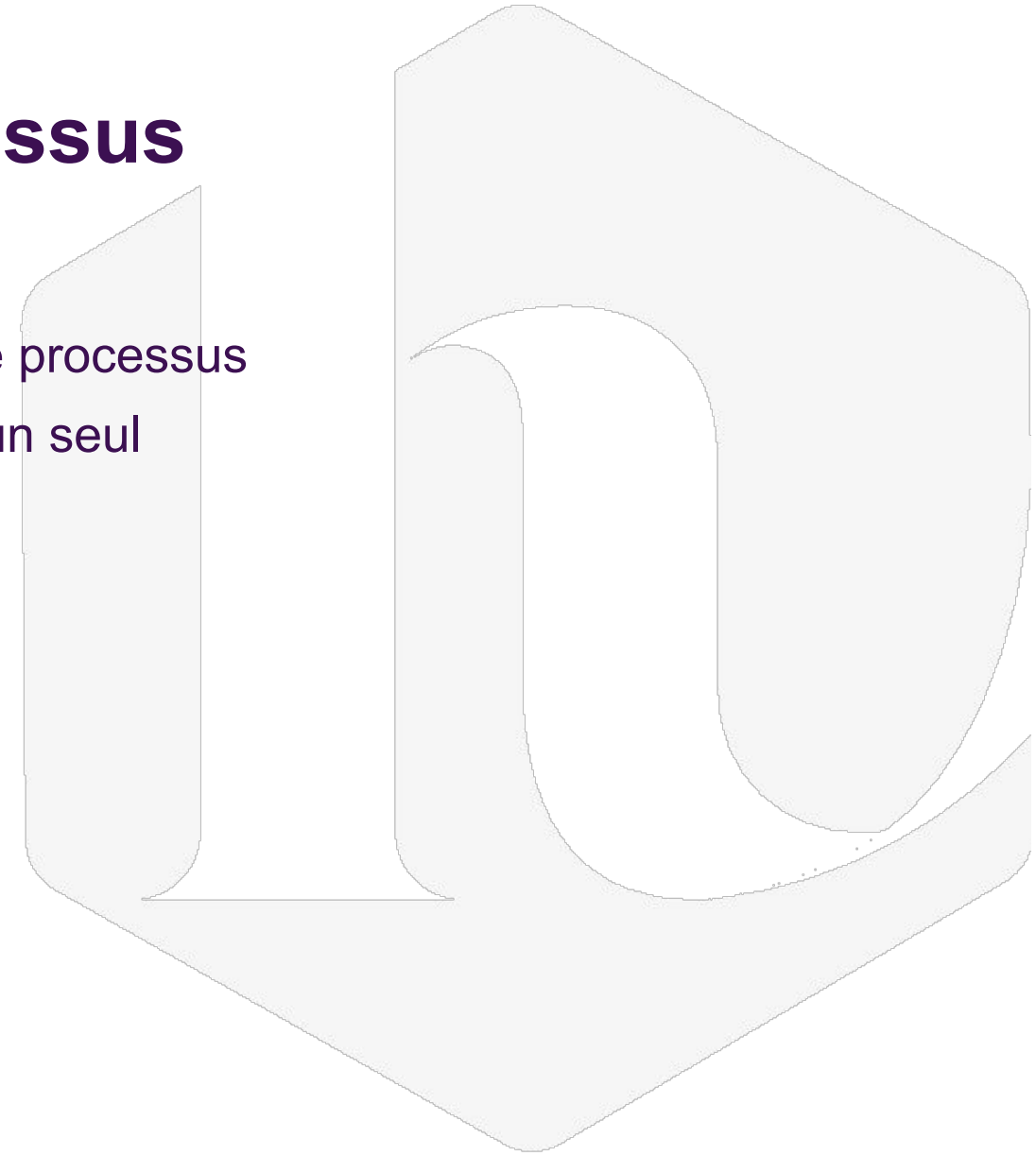
- Génération:
  - Par un événement matériel (division par zéro, etc.)
  - Par un utilisateur (Control-C dans un terminal, etc.)
  - Par un événement logiciel (redimensionnement d’une fenêtre)
  - Par un processus (volontairement, etc.)
- Destinataire:
  - Un thread d’un processus
  - Un processus
  - Un groupe de processus
  - Tous les processus autorisés
- Entre l’envoi et la réception, le signal est dit pending (en attente)

Destinataire: thread

- Envoi à un thread du même processus
- *pthread_kill*

Destinataire: processus

- Envoi à soi-même, ou un autre processus
- Si multi-thread, réception par un seul
- *kill*, *raise*



Destinataire: groupe de processus

- Envoi à un groupe de processus
- Pris en compte par chaque processus
- *kill, killpg*

Destinataire: tous les processus autorisés

- Envoi aux processus auxquels l'émetteur à le droit d'envoyer un signal (voir plus loin)
- Pris en compte par chaque processus
- *kill*

Contrôle d’accès

- L’envoi d’un signal est autorisé si :
  - L’émetteur est root (euid 0)
  - L’émetteur dispose de CAP_KILL (Linux)
  - Cas spécial pour init (pid 1)
    - Ne peut recevoir de signal que pour les handlers qu’il a déclaré
  - L’émetteur et le destinataire ont une identité commune:
    - L’uid effectif ou réel de l’émetteur est égal à l’uid réel ou sauvé du destinataire
    - Permet d’envoyer un signal à un processus ayant changé d’uid !
  - Le signal SIGCONT est traité spécialement:
    - Peut être envoyé à tout processus de la même session
    - Permet de “réveiller” un processus gelé même s’il a changé d’uid

Traitement des signaux

- Les signaux sont reçu en premier par le noyau
- Chaque tâche (thread) dispose d'un masque
- Le masque indique les signaux bloqués
- Un signal bloqué est mis en attente jusqu'à
  - ne plus être bloqué
  - être attendu explicitement
- S'il n'est pas bloqué, le signal est traité
  - généralement, au prochain scheduling du processus ou thread

Traitement des signaux (2)

- Chaque processus a un tableau d’actions
- Tous les threads partagent donc les actions
- Chaque signal a une action par défaut

Actions

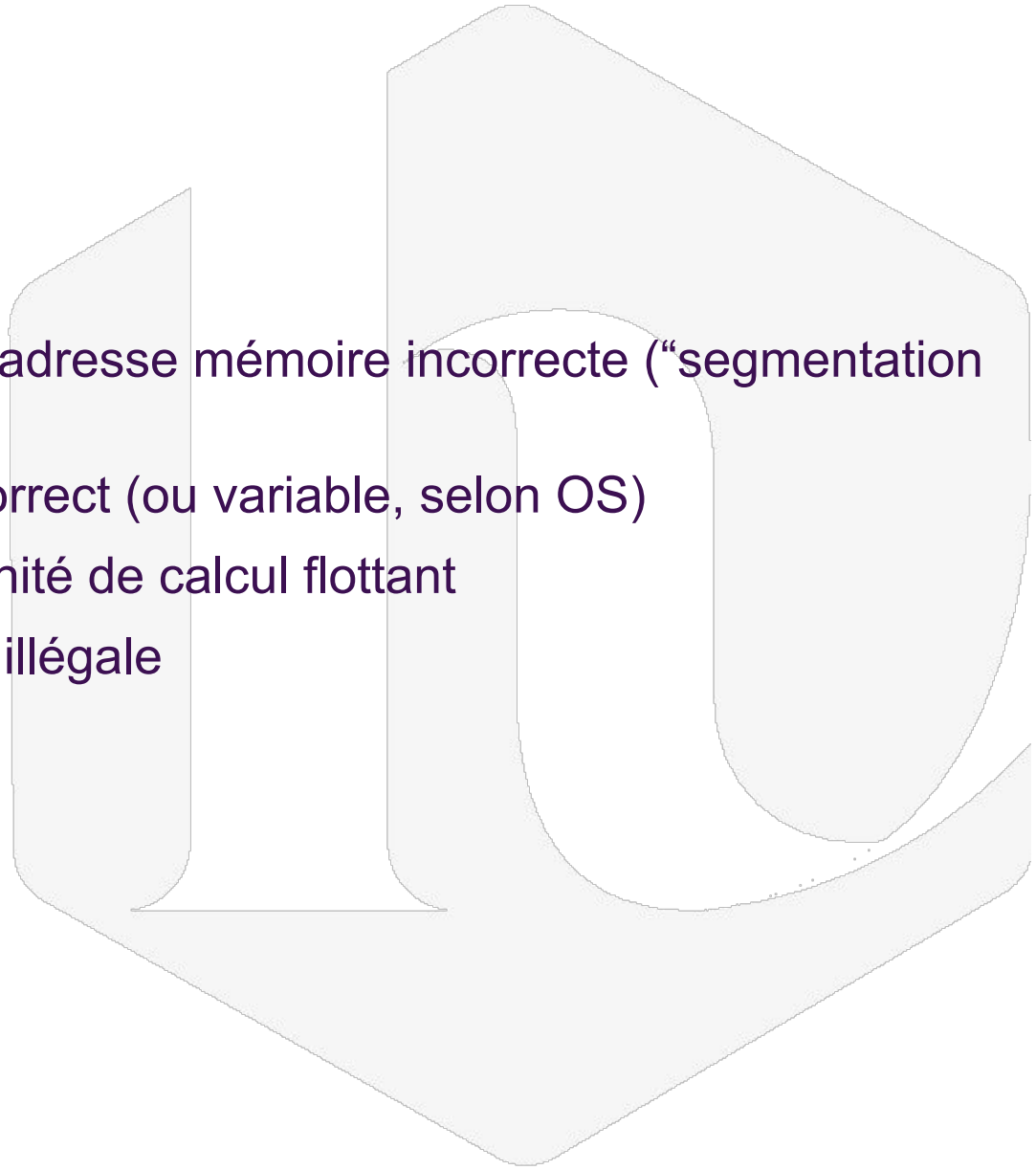
- Si un signal est traité, une action est effectuée:
  - Le signal peut déclencher une routine (handler)
  - Le signal peut être ignoré
  - Le processus peut être terminé, avec ou sans création d'un core dump
  - Le processus peut être gelé
  - Le processus peut être dégelé

Routine de traitement

- À la réception d'un signal associé à une routine:
  - Le destinataire est interrompu
  - Même s'il était dans un appel bloquant
  - La routine est exécutée
  - L'exécution reprend à l'endroit interrompu

Exceptions CPU

- SIGSEGV (sync): accès à une adresse mémoire incorrecte (“segmentation fault”)
- SIGBUS (sync): accès bus incorrect (ou variable, selon OS)
- SIGFPE (sync): erreur dans l’unité de calcul flottant
- SIGILL (sync): instruction CPU illégale



Changements dans l’environnement d’exécution

- SIGCHLD (async): changement d’état d’un processus fils
- SIGHUP (async): déconnexion du terminal de contrôle
- SIGPIPE (async): tube ou socket dont l’extrémité a été fermée
- SIGURG (async): réception en mode urgent
- SIGPOLL (async): événement asynchrone

Signaux liés au terminal de contrôle

- SIGINT (async): demande d'interruption (Ctrl-C)
- SIGTSTP (async): demande de gel (Ctrl-Z)
- SIGQUIT (async): demande d'arrêt (Ctrl-\)
- SIGTTIN (async): tentative de lecture sur un terminal depuis un processus d'arrière-plan
- SIGTTOU (async): tentative d'écriture sur un terminal depuis un processus d'arrière-plan

Contrôle d’exécution du programme

- SIGTERM (async): demande d’arrêt
- SIGKILL (async): arrêt brutal
- SIGSTOP (async): Gel du processus
- SIGCONT (async): Reprise d’un processus gelé

Programmation par la tâche

- SIGALRM, SIGVTALRM, SIGPROF (async): expiration de timers

Signaux de libre emploi

- SIGUSR1, SIGUSR2 (async)

Sauvegarde et restauration du contexte d'exécution

- Une tâche peut sauvegarder son état en un point
- L'exécution continue
- La fonction de restauration peut être appelée plus loin
- Après restauration, la tâche se comporte comme si elle sortait de la fonction de sauvegarde
- Le code de retour permet de différencier les cas

- Non restauré: allocations/libérations, modifications, . . .
- Attention à la pile et aux fonctions !