Problème

Les techniques utilisées

Notre architecture

ghOSt Eramework

Les modèles

57

Conclusion, limites et

perspectives

References

Évaluation des performances des ordonnanceurs c++ et python en userspace avec le framework ghOSt

Présenté par: Armel NGUETOUM

11 Decembre 2023

Plan de la présentation

Introduction

Problème

Les techniques utilisées

Notre architecture

ghOSt Frameworl

Les modèles de scheduling

Résultats

Conclusion, limites et perspectives

- 1 Introduction
 - Contexte
 - Architecture
- 2 Problème
- 3 Les techniques utilisées
- 4 Notre architecture
- 5 ghOSt Framework
- 6 Les modèles de scheduling
- 7 Résultats
 - Les applications CPU Bound
 - Les applications IO Bound
- 8 Conclusion, limites et perspectives References

Contexte: La différence entre l'espace utilisateur et l'espace noyau

Introduction

Contexte Architecture

Problème

Les technique

Notre architecture

ghOSt

Les modèle

Résultats

Conclusion limites et perspective

Reference:

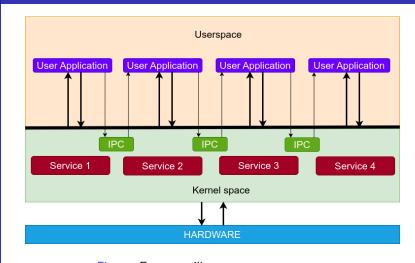


Figure: Espace utilisateur et espace noyau

Contexte: Les Noyaux Monolithiques vs les Micronoyaux

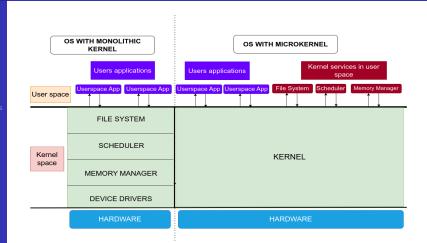


Figure: Noyau Monolithique et Micronoyau

Contexte: La nouvelle tendance des systèmes Linux

Introductio Contexte

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

de schedulin

Conclusion

Conclusion limites et perspective

Reference

Exporter les politiques de gestion des services du noyau en userspace

- Permet le changement de la politique de gestion du service du noyau sans redémarrer l'ordinateur;
- Ouvre le noyau Linux à la communauté informatique avec la réimplémentation du service du noyau en langage de haut niveau;
- Permet d'outrepasser le noyau pour cibler les périphériques depuis l'espace utilisateur;
- Avec le manque des outils de débogage dans le noyau, cette technique va permettre de déboguer le code noyau en userspace.

Contexte: Quelques systèmes qui exportent les services du noyau dans l'espace utilisateur

Introductio

D 110

Probleme

utilisées

Notre architecture

ghOSt Framework

de schedulin

Résultats

Conclusion limites et

- uFs (Liu et al., 2021): Exporte le système de fichier;
- Snap (Michael et al., 2019): Exporte les fonctionnalités réseaux;
- ghOSt (Humphries et al., 2021): Exporte les politiques d'ordonnancement;
- **Demikernel (Irene et al., 2021)**: Outrepasse le noyau Linux en ciblant les périphériques.

Contexte: Architecture des systèmes qui exportent les services du noyau dans l'espace utilisateur

Policy Policy Policy Policy app developer API Library that implement OS service OS service developer User space Kernel space Linux Kernel

Figure: Architecture des systèmes qui exportent les sevices du noyau dans l'espace utilisateur

Contexte
Architecture
Problème
Les technique
utilisées
Notre

ghOSt Framework

Récultati

Conclusion limites et

Les parties importantes de l'architecture

Policy Policy Policy Policy app developer API Library that implement OS service OS service developer User space Kernel space Linux Kernel

Figure: Les parties importantes de l'architecture



Architecture

Problème

Introduction

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

Les modèles de scheduling

Résultats

Conclusion, limites et perspectives

References

Problème

Quelle technique utilisée avec cette architecture pour ouvrir la porte du noyau Linux aux développeurs qui utilisent les langages de haut niveau ?

Première technique : La réécriture du service noyau en langage de haut niveau

Introductior

^Problème

Les techniques utilisées

Notre architectur

ghOSt Framework

Les modèles de schedulin

Récultate

Conclusion, limites et perspectives

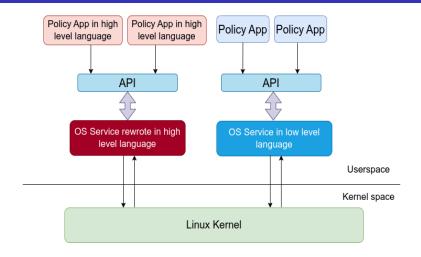


Figure: Réécriture du service noyau en langage de haut niveau

Deuxième technique : Une architecture client-serveur

ntroduction

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

Les modèles

Résultats

Conclusion, limites et perspectives

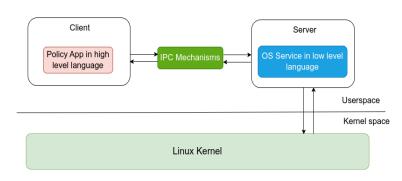


Figure: Architecture client-serveur

Troisième technique : L'utilisation des outils de language binding

Introduction

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

Les modèles

Résultats

Conclusion, limites et perspective

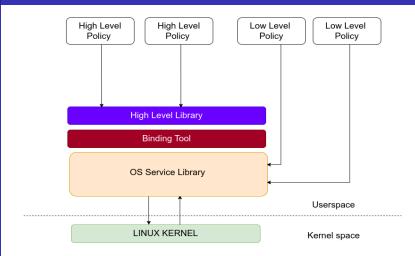


Figure: Les outils de language binding

L'architecture de notre travail

Introduction

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

Les modèles

ne scheduling

Conclusion, limites et

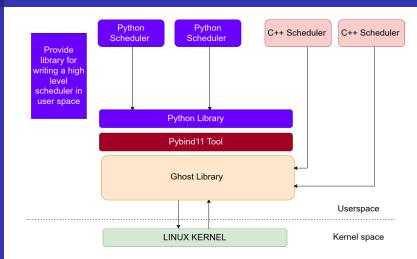


Figure: L'architecture de notre travail

ghOSt (SOSP '21)

Introductio

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

Les modèles de schedulin

Résultats

Conclusion, limites et perspectives

?eferences

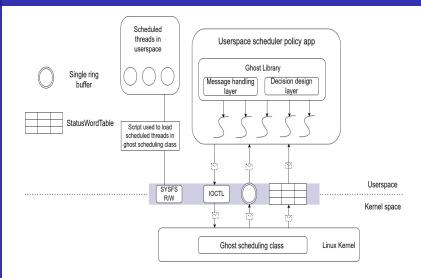


Figure: Présentation de ghOSt

Les modèles d'ordonnancement dans ghOSt

Le modèle centralisé

Introductio

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

Les modèles de scheduling

Résultat

limites et

eferences

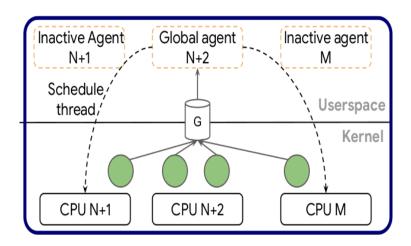


Figure: Le modèle centralisé

Les modèles d'ordonnancement dans ghOSt

Le modèle par cpu

Introductio

Problème

Les techniques utilisées

Notre architectur

ghOSt Eramework

Les modèles de scheduling

Résultat:

limites et

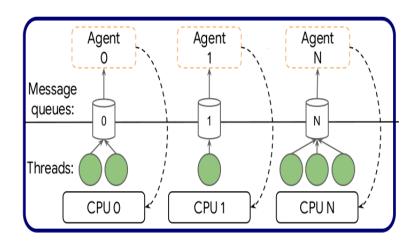


Figure: Le modèle par cpu

Les ordonnanceurs évalués

Introductio

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

Les modèles de scheduling

Résultats

Conclusion, limites et perspectives

deferences

Les ordonnanceurs FIFO (C++ et Python)

- Les ordonnanceurs FIFO avec le modèle par cpu;
- Les ordonnanceurs FIFO avec le modèle centralisé;

Les ordonnanceurs Round Robin (C++ et Python)

- Les ordonnanceurs Round Robin avec le modèle par cpu;
- Les ordonnanceurs Round Robin avec le modèle centralisé;

Problème

Les techniques

Notre

ghOSt

Framework

de scheduling

Résultats

Les applications CPU Bound Les applications IO

Conclusion, limites et perspectives

References

Quelques résultats

Problème

Les techniques

Notre

ghOSt Eramowark

l oc modòloc

de scheduling

Dácultata

Les applications CPU Bound

Les applications IC

Conclusion, limites et perspectives

References

Cas des applications CPU Bound

Temps d'exécution des ordonnanceurs avec 1 thread à scheduler

Introductio

Problème

Les techniques

Notre architecture

ghOSt Framework

Les modèles de schedulin

Résultats

Les applications CPU Bound Les applications IO

Conclusion, limites et perspectives

References

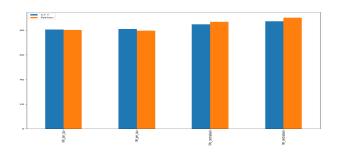


Figure: Temps d'exécution des ordonnanceurs avec 1 thread

■ Les ordonnanceurs FIFO et Round Robin avec le modèle par CPU en python sont meilleurs que ceux en C++.

Temps d'exécution des ordonnanceurs avec 6 threads à scheduler

Introductio

Problème

Les techniques utilisées

Notre architecture

ghOSt Eramework

Les modèles de schedulin

Résultats

Les applications CPU Bound

Conclusion, limites et perspectives

References

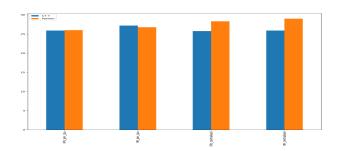


Figure: Temps d'exécution des ordonnanceurs avec 6 threads

■ L'ordonnanceur Round Robin avec le modèle par CPU est meilleur que celui en C++.

Temps d'exécution des ordonnanceurs avec 11 threads à scheduler

Introductio

Problème

Les techniques

Notre architecture

ghOSt Eramowark

Les modèles de schedulin

Résultats

Les applications CPU Bound

Conclusion, limites et

References

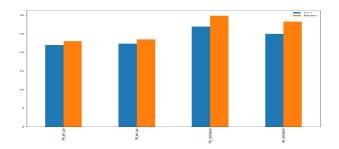


Figure: Temps d'exécution des ordonnanceurs avec 11 threads

■ Les ordonnanceurs C++ sont bien meilleurs que ceux en Python.

Temps d'exécution des ordonnanceurs avec 36 threads à scheduler

Introductio

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

Les modèl

D/- h-r

Les applications CPU Bound Les applications IO

Conclusion, limites et perspectives

Reference

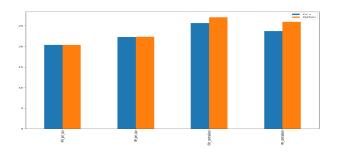


Figure: Temps d'exécution des ordonnanceurs avec 36 threads

 Plus le nombre de threads à ordonnancer augmentent, les ordonnanceurs avec le modèle par CPU deviennent équivalents.

Problème

Les techniques

Notre architecture

ghOSt Framework

Les modèles

de scheduling

Résultats

Les applications CP Bound

Les applications IO

Conclusion, limites et perspectives

References

Cas des applications IO Bound

Temps d'exécution des ordonnanceurs avec 1 thread à scheduler

Introductio

Problème

Les techniques

Notre

ghOSt

Loc modè

_ . .

Les applications CF Bound

Les applications IO Bound

Conclusion, limites et perspectives

References

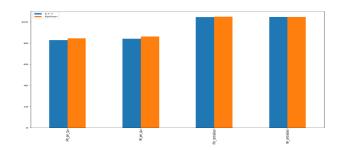


Figure: Temps d'exécution des ordonnanceurs avec 1 thread

 Les ordonnanceurs avec le modèle centralisé sont presque équivalents avec un thread.

Temps d'exécution des ordonnanceurs avec 6 threads à scheduler

Introductio

Problème

Les techniques

Notre

ghOSt Eramework

Les modèle

Résultats

Les applications CF Bound

Les applications IO Bound

limites et perspectives

References

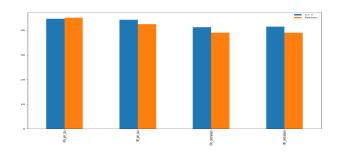


Figure: Temps d'exécution des ordonnanceurs avec 6 threads

■ Avec cette configuration, la plupart des ordonnanceurs python sont bien meilleurs qu'en C++.

Temps d'exécution des ordonnanceurs avec 11 threads à scheduler

ntroduction

Problème

Les techniques

Notre architecture

ghOSt Eramework

Les modèle

Résultats

Les applications CF Bound

Les applications IO Bound

Conclusion, limites et perspectives

References

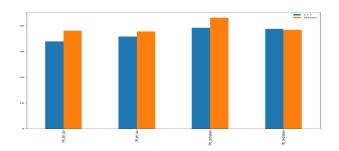


Figure: Temps d'exécution des ordonnanceurs avec 11 threads

L'ordonnanceur Round Robin avec le modèle centralisé en Python est légèrement meilleur que celui en C++.

Temps d'exécution des ordonnanceurs avec 36 threads à scheduler

ntroductio

Problème

Les techniques

Notre architecture

ghOSt Eramework

Les modèle

Résultats

Les applications CF Bound

Les applications IO Bound

Conclusion, limites et perspectives

References

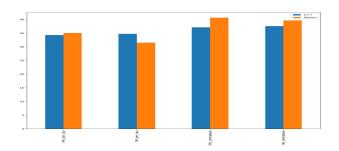


Figure: Temps d'exécution des ordonnanceurs avec 36 threads

■ L'ordonnanceur Round Robin avec le modèle par CPU en python est bien meilleur que celui en C++.

Conclusion

Problème

Choisir parmi les techniques existantes celle-là mieux adaptée pour permettre aux développeurs utilisant les langages de haut niveau d'écrire des politiques de gestion pour les services du noyau exporté en userspace.

Les objectifs

Utiliser la technique des bindings pour proposer une librairie en langage python en utilisant l'outil Pybind11 en se basant sur la librairie du framework ghOSt pour la réécriture des politiques d'ordonnancement en langage python.

B 110

Les techniques utilisées

Notre architectur

ghOSt Framework

Les modèles de schedulin

Conclusion, limites et perspectives

Conclusion

Introduction

Problème

Les techniques

Notre

ghOSt Framework

Les modèles

Dácultata

Conclusion.

limites et perspectives

References

Les résultats

■ En fonction du modèle d'ordonnancement, les politiques d'ordonnancement avec le langage python peuvent surpasser ceux en langage c++.

Limites et perspectives

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

Les modèles de scheduling

Conclusion, limites et perspectives

References

Les limites

Les évaluations n'ont été réalisées que sur un ordinateur portable et non sur un serveur.

Les perspectives

- Instrumentaliser les ordonnanceurs python pour mieux comprendre dans quelles conditions ils sont meilleurs que ceux en C++ (GC et GIL);
- Écrire des politiques d'ordonnancement complexes en Python;
- Faire du binding sur d'autres langages de programmation comme Rust, Go.

References I

References

Axboe, J. (2019). liburing code [Computer software manual]. Retrieved 20/01/2023, from https://github.com/axboe/liburing

Humphries, J. T., Natu, N., Chaugule, A., Weisse, O., Rhoden, B., Don, J., ... Kozyrakis, C. (2021, October). ghost: Fast and flexible user-space delegation of linux scheduling. In Acm sigops 28th symposium on operating systems principles (sosp '21) (pp. 588–604). ACM, New York, NY, USA. (Virtual Event, Germany.) doi:

http://dx.doi.org/10.1145/3477132.3483542

References II

Introductio

Problème

Les techniques utilisées

Notre architecture

ghOSt Framework

Les modèles de scheduling

Résultats

limites et perspectives

References

Irene, Z., Amanda, R., Pratyush, P., Kirk, O., Jacob, N., Omar, N. L., ... Anirudh, B. (2021, October). The demikernel datapath os architecture for microsecond-scale datacenter systems. In Acm sigops 28th symposium on operating systems principles (sosp '21). ACM, New York, NY, USA. (Virtual Event, Germany.) doi: http://dx.doi.org/10.1145/3477132.3483569

Kerrisk, M. (2022). proc(5) — linux manual page [Computer software manual]. Retrieved 10/04/2023, from https://man7.org/linux/man-pages/man5/proc.5.html

References III

mtroductio

Problème

Les techniques utilisées

Notre architectur

ghOSt Frameworl

Les modèles de scheduling

Dácultata

Conclusion, limites et perspectives

References

Liu, J., Rebello, A., Dai, Y., Ye, C., Kannan, S., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2021, October). Scale and performance in a filesystem semi-microkernel. In *Acm sigops 28th symposium on operating systems principles (sosp '21)* (pp. 819–835). ACM, New York, NY, USA. (Virtual Event, Germany.) doi: http://dx.doi.org/10.1145/3477132.3483581

Michael, M., Marc, d. K., Jacob, A., Christopher, A., Sean, B., Carlo, C., . . . Amin, V. (2019, October). Snap: a microkernel approach to host networking. In *Acm sigops 27th symposium on operating systems principles (sosp '19)* (pp. 399–415). ACM, New York, NY, USA. doi: http://dx.doi.org/10.1145/3341301.3359657

Problème

Les techniques utilisées

Notre architecture

ghOSt Eramowark

Les modèles

Résultats

. . .

limites et perspective

References

MERCI POUR VOTRE AIMABLE ATTENTION