ROOP：ベアメタルプログラム向け オンラインジャッジプラットフォーム

西村啓佑[1,2,a)]　粟本真一[1]　千代浩之[1]　加藤真平[1]

概要：本論文では，ベアメタルで動作するプログラムを対象にしたオンラインジャッジプラットフォーム ROOP（Ring-0(O) Online judge Platform）を提案する．ROOP は Web インターフェースを備え，それを経由して問題に解答するとすぐに正誤の判定を試み，結果を Web で通知する．これにより大学の OS の講義に導入することで課題の採点補助や学生が試行錯誤を補助するといった活用や，OS 分野を対象としたプログラミングコンテストの開催が可能になった．実際に提案システムは大学の講義演習で使用され，講義後のアンケートでは 19 名中 17 名が使いやすかったとの回答を得た．ROOP は提出されたコードをコンパイル及び実行する際に Docker や QEMU を使用ポータビリティとセキュリティを向上させている．これらは大量にメモリを消費するため，システムの安定動作のためにはメモリリソースの管理が必要となる課題が存在するが，提案システムにおいて同時に起動する Docker コンテナや QEMU のインスタンスの数を制御することでそれを解決した．また，Docker や QEMU の起動にはオーバヘッドが生じるため，コードが提出された後にそのような処理を行うと，ジャッジ完了までの待ち時間が長くなってしまうという欠点がある．そこで，ROOP は全ての解答に対して必要な処理を抽出し，予めコードが提出されるまでに終わらせることで，正誤判定の待ち時間を最大 28%短縮した．

キーワード：OS 教育, ベアメタルプログラム，オンラインジャッジ, 仮想化技術

1. はじめに

OS は現代のコンピュータ技術に不可欠な存在であり，コンピュータサイエンスを学ぶ上で重要である．大学の OS の授業では，Nachos[1] や OS161[2] といった教育用 OS を用いたり，Linux のカーネルを用いた演習 [3], [4] などを通して，ベアメタルで動作するソフトウェアについて手を動かしながら学んでいくという形式が採用されているものの，ベアメタルプログラミング特有の学習コストの高さは依然として存在し，より良い学習モデルの提案が待望されている．なお，ここでいうベアメタルとは x86-64 における Ring-0，aarch64 における例外レベル 1 で動作することを指す．

大学のアルゴリズムの授業では，学生の解答にたいする評価を迅速かつ正確に行うため，オンラインジャッジシステムが用いられる事がある [5], [6], [7]．この場合，参加者はジャッジシステムに設定された要件を厳密に満たさなければ一方で，評価結果を瞬時に得る事ができるため，学生は何度も試行錯誤の中で解答を導き出すことができる．また，オンラインジャッジの導入は，教員にとっても部の課題の採点が自動化できるメリットがある．オンラインジャッジシステムを活用した講義演習の流れは，以下の通りである．まず，教員側がいくつかの問題を出題し，参加者がその解答を作成する．学生はオンラインジャッジシステムに対して解答を提出し，ジャッジサーバ上で解答の正当性を検証された後，正当性に応じてスコアリングが行われる．教員と学生は，それぞれの正解した問題に応じて習熟度を確認することができる．

オンラインジャッジは，昨今様々な分野で流行しているコンピュータの技能を測ることを目的としたプログラミングコンテスト，特にアルゴリズム実装能力を競う競技プログラミング等で利用されることもある [5], [8], [9], [10], [11]．プログラミングコンテストは様々な分野を対象にしたものがあるが，それらは参加者の競争心を刺激し，参加者による自発的な競技分野全体の盛り上げに繋がる．参加者はコンテスト期間にとどまらず，次のコンテストへの準備として，日々鍛錬を重ねるからである．また参加者間で，Twitter

[1] 東京大学
The University of Tokyo, Bunkyo, Tokyo 113-8654, Japan
[2] 東京工業大学
Tokyo Institute of Technology, Meguro, Tokyo 152-8550, Japan
[a)] keisuke.nishimura@pf.is.s.u-tokyo.ac.jp

やブログ等の SNS 上での情報公開が盛んに行われる結果，初学者がその分野に参入しやすい土壌も構築される．

著者らは OS 分野を対象としたオンラインジャッジを提案する．これにより，OS の講義にオンラインジャッジを導入することが可能になり，この分野の新たな学習モデルの形成につながる．さらに，OS を対象としたプログラミングコンテストの開催も可能となり，OS 分野全体の盛り上げに貢献できることなどの効果が期待できる．

OS 分野を対象にしたオンラインジャッジが可能性を秘めている一方，その開発は容易ではない．要因の一つとして，ユーザ空間で動くソフトウェアを対象とした他のプログラミングコンテストと異なり主催者側での検証に手間を要するため，提出されたコードの実行が難しいことが挙げられる．特に，オンラインジャッジシステムの導入に当たっては，ジャッジサーバ上でベアメタルで動作するプログラムをどのように実行するか，という点を解決する必要がある．このような問題があるため，既存のオンラインジャッジシステムをベアメタルで動作するソフトウェアに適用することはできない．

そこで著者らは，OS 分野向けのオンラインジャッジの開発，運用する上で生じ得る問題を分析し，それらの問題を解決したオンラインジャッジプラットフォームである ROOP（Ring-0(O) Online judge Platform）を開発した．ROOP は従来のオンラインジャッジでは不可能だったベアメタルプログラム，x86-64 アーキテクチャにおける Ring-0 で提出されたプログラムを実行，検証する事ができる．

ROOP は提出されたコードをコンパイル及び実行する際に Docker [12] や QEMU [13] を使用することで，ポータビリティとセキュリティを向上させているが，それらの起動にはオーバヘッドが生じるため，コードが提出された後にそのような処理を行うと，ジャッジ完了までの待ち時間が長くなってしまうという欠点がある．そこで，ジャッジ前に予めそれらを起動しておくことで正誤判定の待ち時間を短縮した．

また，Docker や QEMU は多くのメモリを消費する傾向があるため，システムの安定動作にはメモリリソースの管理が重要な課題になる．そこで同時に起動する Docker コンテナと QEMU の数を制御する機構を実装することで，この課題を解決した．

ROOP は，実際に東京大学理学部情報科学科の授業内で活用された．これにより従来よりも演習課題の解答と提出プロセスを簡略化され，学生と指導側の負荷を減らす事に成功した．事実，授業終了後に実施したアンケートの結果では，学生が ROOP に対して高い満足度を示したことがわかった．

本研究の貢献は以下の通りである．

- OS 分野を対象としたオンラインジャッジシステムに必要な機能や生じうる問題を分析し，ROOP を開発した．
- 実際に授業で提案システムを活用し評価した．
- 提案システムが大量にメモリを消費することを抑制し，システムの安定な動作を実現した．
- 提案システムがジャッジにかかる時間を短縮した．

本論文の構成は以下のようになっている．2 章でオンラインジャッジに関する研究事例を紹介する．3 章では，ROOP の設計と実装について説明する．4 章では ROOP の評価として，大学の演習で使用された際のアンケート結果やジャッジにかかる時間の性能について述べる．最後に 5 章で本研究をまとめる．

2. 関連研究

2.1 オンラインジャッジ

オンラインジャッジとは，ユーザから提出されたプログラムをコンパイル及び実行し，厳格な検証データや検証器を用いて，その正確さと性能を自動評価しフィードバックするサービスと定義することができる [10]．本論文ではユーザが提出したプログラムのソースコードを「サブミット」，プログラムの評価結果を「ジャッジステータス」と呼ぶ．

オンラインジャッジのサービス提供するためには，問題とその解答を作成するだけでなく，オンラインジャッジプラットフォームと呼ばれるユーザにインターフェースを提供するフロントエンドや実際にジャッジをするバックエンドを実装する必要がある．そのの多くはクローズドソース [14] だが，DMOJ [15] や Lutece [16] など OSS として公開しているものもある．

従来のオンラインジャッジは多くがアルゴリズムの良し悪しを競う競技プログラミングに使用されているため，評価対象のプログラムは Ring-3 のユーザ空間で動作することが前提となっている．競技プログラミングの例として，ICPC [9] や AtCoder [11]，Aizu Online Judge [10]，UVa Online Judge [8]，Sphere Online Judge [5] 等が挙げられる．なお，筆者らが探した限り Ring-3 以外で動作するプログラムを対象としたオンラインジャッジは見つけることができなかった．

オンラインジャッジはしばしば教育を目的として利用される．例えば，Jutge.org [17] は教育を目的としたオープンアクセスのオンラインジャッジが成功した例である．また，大学の講義や演習の一環として導入される例もある．その目的の一つとして，BOSS [6] を始めとするプログラムの自動検証 (Automated Assessment) を用いた学生が提出したプログラムの正誤判定の自動化があげられる．シンガポール国立大学で授業にオンラインジャッジを導入したところ，教員がより多くの宿題を出すことが可能になったという報告がある [7]．

2.2 OS 分野の教育

OS の実装に関する教育を目的としたプロジェクトは数多く存在するが，なかでも教育用に実装されたハードウェアを用いるアプローチがある．例えば，Nachos [1] や OS 161 [2] で利用されているハードウェアは教育用に設計されている．理想的なシステム上で動作することを仮定しているため，学生は細かなハードウェアの仕様に煩わされないでよいが，実際の OS 開発に即していないという側面もある．

一方で，x86 をはじめとする現実のアーキテクチャを対象とした OS も教育に使用されている．GeekOS [18] や Pintos [19] は x86 上で動作することを想定しており，学生は Bochs 等のエミュレータを通して開発を行う．また，Nieh [3] らや Hess [4] らによる大学の OS の授業に汎用 OS である Linux を活用したという報告もある．

3. システム設計

3.1 出題

ROOP が出題の対象としているのは OS 分野, 特に Ring-0 で動作するソフトウェアに関する問題であり，従来のオンラインジャッジが出題の対象としている数学的な問題とは性質が異なる．以下では，ROOP が扱うことができる問題をデバイス制御とアルゴリズムの 2 つに分類し，それぞれの特徴と出題方法について説明する．

3.1.1 デバイス制御

ROOP では NIC やタイマなどのデバイスの制御に関する問題を出題することができる．例えば，現在多くのパーソナルコンピュータでタイマの一種として使用されている HPET の制御について出題が可能である．HPET は通常 ACPI を通じて取得する特定のメモリアドレスにマッピングされたレジスタを通じてアクセスすることができる．そこで，提案システムの例題として，次のような問題 1 が考えられる．

問題 1 デバイス制御 (HPET)

HPET のレジスタのメモリアドレスを入力し，レジスタ情報の一部を出力するプログラムを作成せよ．

デバイス制御を学びたい初学者がオンラインジャッジを使用することは，対象デバイスのドライバを自作するだけに比べて学習効率が高いと考えられる．なぜなら，オンラインジャッジは対象デバイスに直接関係ないが依存している技術を取得する手間がかからないからである．問題 1 では本来 ACPI の制御を行って HPET レジスタのメモリアドレスを取得する必要があったが，出題者が予め ACPI の制御を行ってメモリアドレスを取得しているためユーザがその実装を行う必要はない．

また，一般的なオンラインジャッジと異なり解答方法として「穴埋め形式」を採用した．デバイスの制御の問題の入出力は，特定のメモリアドレスを読み書きすることで行う．しかしながら，Ring-0 のアプリケーションは不正なメモリを参照することで PC がクラッシュしてしまうことがありデバッグが困難である．そこで，この入出力などの前処理，後処理はあらかじめ出題者が実装しておくことで，ユーザの些細なミスで問題が不正解とジャッジされることを防ぐこととした．

3.1.2 アルゴリズム

ROOP はロックやセマフォ等の排他制御に関するアルゴリズムの出題も想定している．これは現代のパーソナルコンピュータの多くはマルチコアで動いており，OS はコア間の調停をする必要があるからである．例えば，次のような問題 2 が考えられる．

問題 2 アルゴリズム (ロック)

関数 Func() は 4 つのコアから同時に複数回呼ばれ，一つのグローバル変数を読み書きする．
Func() の先頭と末尾にある lock()，unlock() 関数を実装することで Func() が適切な動作ができるようせよ．

問題 2 を含む，多くのアルゴリズム問題も前項で説明した「穴埋め形式」を採用することを想定している．先の例では，関数 Func() の実装やマルチコアで動かすためのデバイス制御を予め出題者側で行っている．

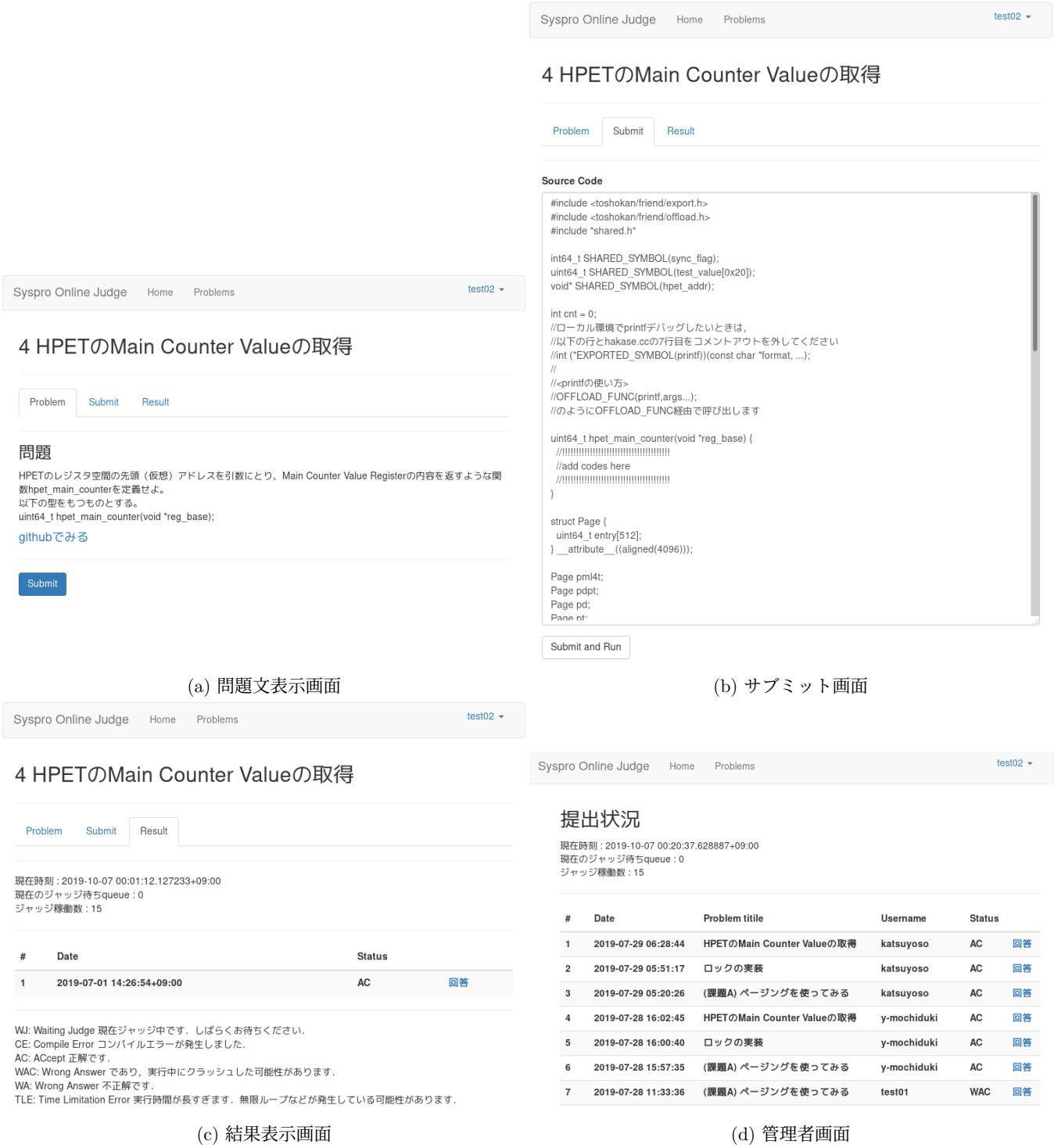
問題 2 は関数 Func() を実際にマルチコア環境で 10,000 回程度呼び出し，不整合が発生していないか確認することで正誤判定をおこなった．例えば，ロックやアンロックの実装が不十分な場合，無限ループに陥ったり不正なアドレスにアクセスしてクラッシュしてしまう．この判定方法では必要条件しか確認できないが，実際の出題では誤検知や検出漏れなどは発生しなかった．

3.2 インターフェース

ROOP は一般的なオンラインジャッジと同様に問題の閲覧（**図 1a**），解答の送信（**図 1b**），ジャッジステータスの確認（**図 1c**）といった操作が可能な Web ページを提供する．また，管理者用に全てのサブミット提出状況を確認するページや問題の追加や修正を行うためのインターフェースも提供している．

問題の管理 出題する問題の追加は規定のフォーマットに沿ったファイルを作成し，指定のディレクトリに配置することで行う．この設計により，問題の管理に git 等のバージョン管理システムを使用することが可能になる．また，問題文のフォーマットには Markdown 記法を採用しており，ROOP 以外の Markdown ビューワが使える様々な環境で閲覧，編集が可能である．

管理者画面（図 1d） 規定のユーザには管理者としての権



(a) 問題文表示画面　(b) サブミット画面

(c) 結果表示画面　(d) 管理者画面

図 1: ROOP の Web インタフェース

限が与えられており，提出されたサブミットの詳細をリアルタイムで確認することができる．この機能は授業の演習などにこのシステムを利用する際に使用されることを想定しており，出題した問題の難易度の把握や学生の理解度に応じた柔軟な授業の実現に利用される．

ジャッジステータス ROOP はジャッジステータスとして **表 1** に示す 8 つに対応している．このうち，WAC 以外は UVa Online Judge [8] をはじめとする既存のシステムを参考にした．WAC は Wrong Answer Crash の頭文字をとったもので，実行時にクラッシュしたことを意味しており，ユーザランドでの動作を仮定した従来のオンラインジャッジにはなかったステータスである．なお，既存のシステムにはランタイムエラーを意味する RE というステータスが存在するが，これは OS によって予め決められた特定の挙動がトラップされることによって発生するもので，マシンのクラッシュとは異なるものである．

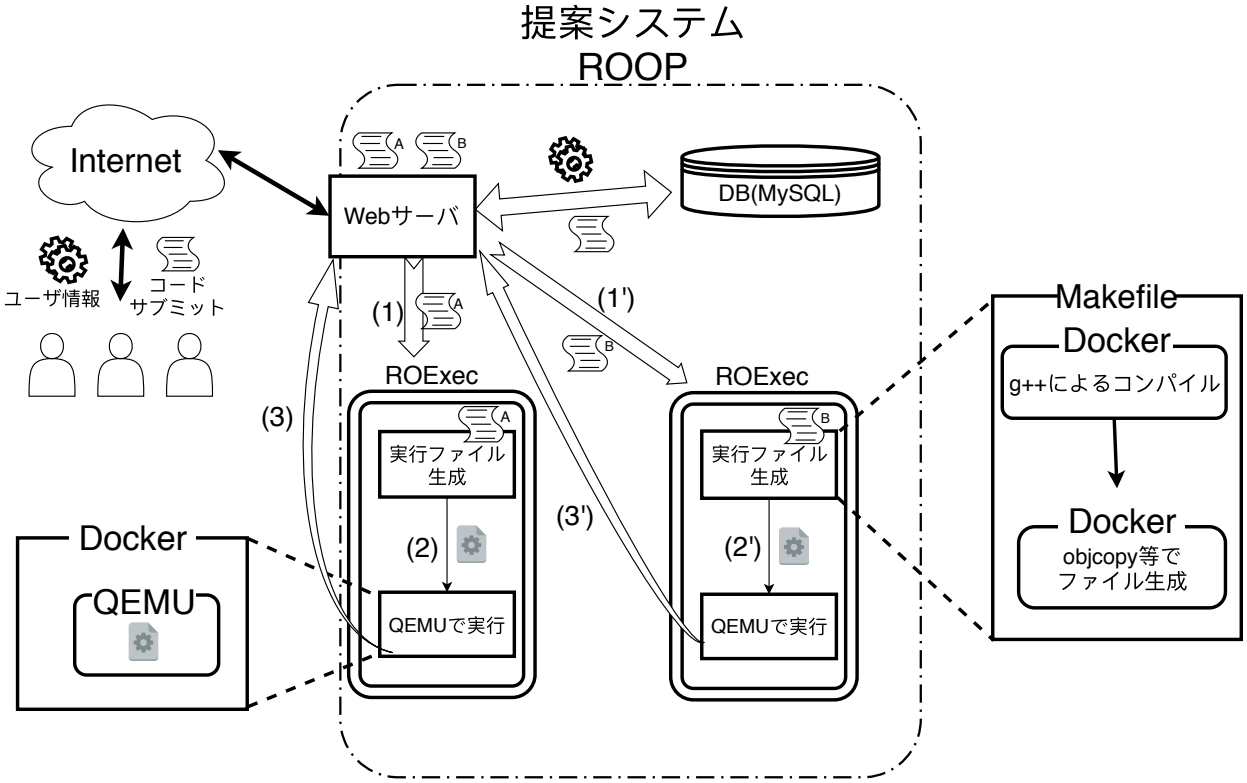


図 2: ROOP アーキテクチャ

表 1: ROOP のジャッジステータス

	頭文字	意味
AC	ACcept	正解
WA	Wrong Answer	最後まで実行できたが不正解
WAC	Wrong Answer Crash	途中でクラッシュして不正解
TLE	Time Limitation Error	規定の実行時間をオーバー
CE	Compile Error	コンパイルに失敗
WJ	Waiting Judge	ジャッジ待ち
IE	Internal Error	サーバ内部のエラー

表 2: ROOP のバージョン

	バージョン 1(v1)	バージョン 2(v2)
目的	授業での使用（4.2.1 節）	バージョン 1 の性能向上
高速化	無し	有効/無効 切替可能

3.3 アーキテクチャ

提案システムのアーキテクチャの概要を説明する．**図 2** に示すように，このシステムは Web サーバと ROExec と呼ばれる 2 つのコンポーネントに分かれている．Web サーバは 3.2 節で説明したインターフェースをユーザに提供する．これは Python とそのフレームワークである Flask で書かれており，Nginx と uWSGI を用いてデプロイされる．ROExec は Python とビルドシステムとして Makefile を使って実装されており，サブミットを受け取った Web サーバによって起動されるコンポーネントで，サブミット一つにつき一つのインスタンスが対応し，コンパイルと実行ファイルの生成および実行までを行う．そのため，複数の ROExec が同時に動作することがあるが，多くの Docker コンテナや QEMU のインスタンスが生成されてしまい，メモリが枯渇してしまうことを防ぐためにその数には上限を設定した（4.2.2 項参照）．コンパイルエラー等でうまく実行できなかった場合も含めて，ROExec は必ず一つ WJ 以外のジャッジステータスを Web サーバに返す．ROOP ではサブミットのコンパイルと実行を行う際に，ポータビリティ向上のために Docker を利用している．

また，ROOP は **表 2** に示す 2 つのバージョンがある．どちらも図 2 で示したアーキテクチャとインターフェースは同じだが，バージョン 2 はバージョン 1 の授業での使用経験（4.2.1 項参照）経験を踏まえたうえで，性能向上を目的として再実装したものである．

3.3.1 実行フロー

図 2 を元に説明する．Web サーバにサブミットがあると ROExec に渡され (1)，Docker 上で動く g++によって

コンパイルが試みられる．このとき，ジャッジステータスを WJ に設定する．もし文法エラーなどでコンパイルが上手く行かなかったときは ROExec はジャッジステータス CE を返し，Web サーバがそれを受け取ってステータスを更新する．コンパイルに成功すると，そのバイナリから実行可能ファイルを生成する．このファイルは特殊な ELF ファイルで，後述する Linux によって実行時にメモリに展開される．なお，実行可能ファイルの生成に失敗した場合，ジャッジステータス IE が返される．

実行可能ファイルが生成されると，Docker 上の仮想マシン内にコピーされ QEMU を用いて実行される (2)．仮想マシンでの実行中に GP 例外の発生等でコードがクラッシュした場合，それを検出した ROExec はジャッジステータス WAC を返し終了する．正常終了した場合には仮想マシンが AC/WA をジャッジして，ROExec がそれを受け取り Web サーバに通知する．

ROExec から通知を受け取った Web サーバは (3)，DB のジャッジステータスを書き換え Web ページにそれを反映する．以上でジャッジは全て完了となる．

なお，ROExec から呼び出す他のアプリケーションは全て Docker コンテナ上で動作するため，Docker さえインストールしてあればどのような環境でも ROOP の実行が可能になる．

3.3.2 高速化

ROOP の Ring-0 コード実行部分に使用されている QEMU や, コンパイル及び実行時に使用される Docker コンテナ群の起動や終了には時間がかかる．そこで, ROOP(v2) では，サブミットを受け取ってからジャッジが完了するまでの時間を短縮するために，次の機能を有している．

(1) サブミットが到着する前に Docker コンテナと QEMU を起動させておく．
(2) ジャッジステータスが確定した時点で ROExec は Web サーバに結果を通知し，Docker コンテナの終了処理は後で行う．

高速化機能の有効/無効の切り替えや予め起動させておく Docker コンテナと QEMU の個数の設定は ROOP(v2) の起動時に行うことができる．なお，高速化機能による性能の変化は 4.2 節で説明する．

4. 評価

4.1 授業での使用

提案システム ROOP(v1) は 2019 年度東京大学理学部情報科学科のシステムプログラミング実験の一部 (2019/6/17-29) で使用された．講義内では 3.1 節での検討を踏まえた問題 5 問を出題し，講義の受講者数は合計で 35 人，そのうち実際に ROOP を使用した人数は 33 人だった．なお，システム使用者 33 名については本研究の趣旨をご理解頂き，アンケート内容や解答の一部を本研究で活用することに同意して頂いている．講義期間中の合計サブミット数は 194 回だった．

講義終了後の学生に対し，ROOP のユーザーエクルペリエンスに関する任意提出の匿名アンケートを行い，19 名からの回答を得た．「オンラインジャッジの Web ページは使いやすかったか？」という質問では，「当てはまる」または「よく当てはまる」と答えた受講者の割合は 89.5%にのぼった．このことから，ROOP の Web インターフェースがユーザーに好意的に評価されていることがわかった．一方で，コードをサブミットするためのテキストボックスをシンタックスハイライトした方が良いという意見があった．これは今後の課題としたい．また，ROOP(v1) では最終的に AC とジャッジされるサブミットに対するジャッジ時間の平均は 33 秒だった．これについて「ジャッジにかかる時間は問題なかった.」という質問には，全員が「当てはまる」または「よく当てはまる」と答えた．したがって，30 人規模の講義ではジャッジ時間の性能の面では問題なく ROOP を使用することが可能であると考えられる．

また，本講義の TA2 人にも聞き取り調査をおこなった結果，次のような意見があった．

(1) ブラウザ上で演習ができるという点で気楽に動かせるのが良い．
(2) OS の内部挙動をよく理解しないまま，根拠のない場当たり的な修正を加え続けることで正解にたどり着く学生がいた．
(3) ROOP は学生の解答を採点する補助になった．

従来の OS 教育に使用されていたシステムは，まず学生が使用するコンピュータに課題の動作環境をセットアップしする必要があったが，ROOP ではその必要はなくブラウザさえ動けばよい．したがって，(1) での指摘の通り学生はより手軽に演習に着手できたと考えられる．一方で，ブラウザ上で簡単に正誤が確認できるがゆえに (2) のような学生が出てしまったという指摘があったが，演習問題の質をあげ本当に理解していないと解けない問題を出題することで回避できると考えている．また (3) の指摘の通り，オンラインジャッジの導入によってとで人間が手作業で採点する箇所が減ったため, TA にとって採点はより簡単になったと考えられる．

4.2 ジャッジ時間

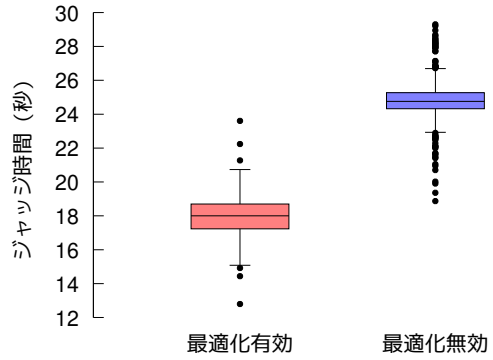
以下では，ROOP(v2) の性能を評価するために**表 3** に示すコンピュータ上で動作させた．

4.2.1 負荷がない状態での性能

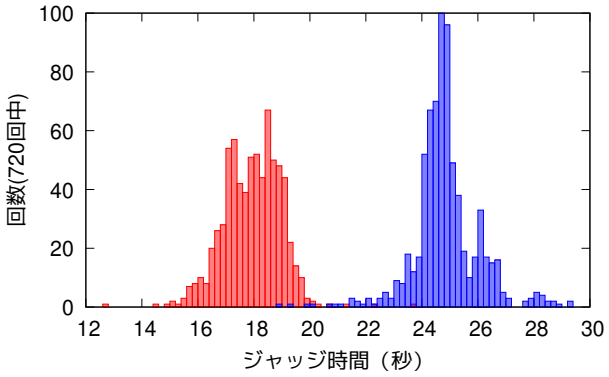
負荷がない状態，つまりあるサブミットをジャッジ中に他のサブミットが無い状態でなるジャッジ時間性能を計測した（**図 3**）．このサブミットは，節で使用した問題に対する回答で最も多かった，ジャッジステータスが AC となるものを利用した．

表 3: ROOP の評価に用いたコンピュータ

プロセッサ	Intel(R) Xeon(R) CPU E5-2650 v2
動作周波数	2.6GHz
コア数	4
メモリサイズ	8GB
OS	Linux (Ubuntu)



(a) ジャッジ時間の箱ひげ図



(b) ジャッジ時間のヒストグラム

図 3: ROOP(v2) ジャッジ時間（負荷なし）

ジャッジにかかる時間の平均は高速化が有効だと 17.9 秒で無効だと 24.8 秒だったことから，約 28%（6.9 秒）の高速化が実現できた．ROOP で使用している Docker コンテナの起動にかかる時間の平均が 5.7 秒，終了にかかる時間の平均が 1.8 秒程度だったことから，3.3.2 項で説明した手法にはオーバヘッドはほとんど無いことがわかる．

4.2.2 負荷がある状態での性能

本項では ROOP(v2) に対して 4.2.1 項と同じサブミットを 4 秒に 1 度 30 回送信して負荷をかけた状態でジャッジ時間を計測する．

はじめに，ROExec 同時起動上限数を十分大きくして ROOP のジャッジ時間を計測したが，高速化の有無に関わらず 30 個のサブミットのうち約半数のジャッジがサーバ内部のエラーで一度失敗したためメモリ使用量を調査した（**図 4**）．その結果，空きメモリが減ってスワップが大量に発生しており，失敗したジャッジで使用した QEMU のログを調べると *cannot set up guest memory 'pc.ram':*

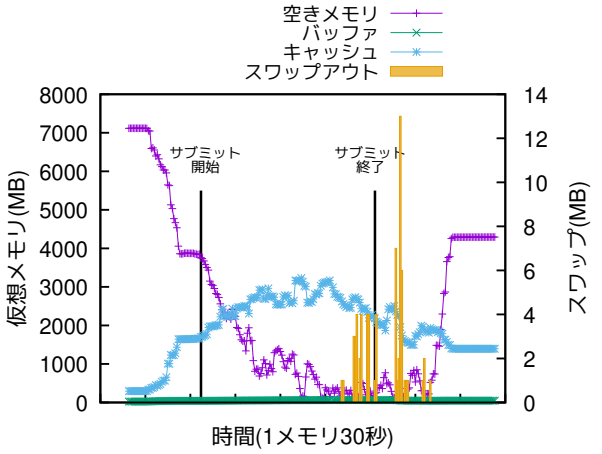


図 4: ROExec 同時起動上限数が十分大きなときのジャッジ時間計測時メモリ使用量

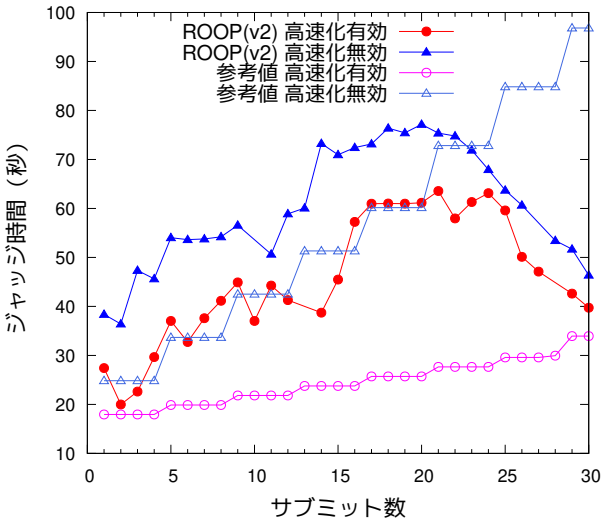


図 5: ROOP(v2) ジャッジ時間（負荷あり，ROExec の同時起動上限数:8）

Out of memory といったエラーで終了していたことから，ジャッジの失敗は QEMU や Docker コンテナを大量に起動したことによりメモリを使い果たしたことに起因すると考えられる．

続いて，ROExec 同時起動上限数を 8 と十分小さくして高速化を有効にした ROOP と高速化を無効にした ROOP の 2 つでジャッジ時間の比較計測をおこなった（**図 5**）．なお図には，1 つあたりの処理時間が**図 3a** の平均値で窓口が CPU のコア数と同じ 4 つの待ち行列モデルを想定した参考値を記載している．その結果，それぞれメモリ不足で QEMU が起動できなかったことによるサーバ内部のエラー発生回数は 30 回中 1-2 回程度に抑えることができた．エラーの数を 0 にすることは出来なかったが，全体に占める割合が数%なのでこのエラーを検出した際に再ジャッジを行うことで対応ができると考えている．

図 5 をみると，高速化した場合もそうでない場合も負荷

がない状態と比べてサブミット数が 20 付近までジャッジ時間が増加してゆき，その後は減少してゆく．これはサブミット数 20 近辺で 30 個全てのジャッジが到着し，それ以降はサーバに対する負荷が減少することに起因すると考えられる．また，高速化を無効にした場合の後半を除いて全てのジャッジ時間は参考値より長くなった．これは参考値は負荷がない状態で計測したジャッジ時間をもとに算出されており，負荷状態ではジャッジ時間が長くなってしまったからだろう．一方，高速化を無効にした場合の後半が参考値より短くなったのは，新たなサブミットがない分負荷が小さくなっている点に加えて，大量に同じソフトウェアの起動と終了を繰り返したことによりキャッシュの影響が大きくなり高速化につながった可能性がある．

また，今回の実験では計測出来なかったが，ROExec の同時起動上限数を大きくしていった場合，高速化のために予め Docker コンテナや QEMU を起動しておくためのリソースが不足し，高速の影響は小さくなって行くことが予想される．

5. まとめ

本論文では Ring-0 ソフトウェアを対象としたオンラインジャッジ ROOP を開発し，既存のオンラインジャッジでは不可能だった Ring-0 で動くプログラムをジャッジすることが可能になった．ROOP の実装において，(a) 内部で使用している Docker コンテナや QEMU の影響で前処理や後処理に時間がかかる，(b) それらのアプリケーションが大量のメモリを消費してしまうことでシステムが不安定になるという 2 つの課題がみつかった．しかしそれらは，(a) については予めその処理をおこなっておくことでサブミットしてから結果がでるまでの間の時間を最大 28% 短縮することで，(b) については同時に起動する Docker コンテナや QEMU のインスタンスの数を制御することで解決できた．ただし，ジャッジの失敗を 0 にすることはできなかったので今後の課題としたい．また，実際に東京大学理学部情報科学科での授業に使用した結果，提案システムの Web インタフェースについても満足度が高い結果が得られた．今回は x86-64 アーキテクチャを対象とした問題にのみ対応していたが，今後は aarch64 を始めとする他のアーキテクチャにも対応していきたい．

謝辞 この成果の一部は，国立研究開発法人新エネルギー・産業技術総合開発機構（ＮＥＤＯ）の委託事業の結果得られたものです．また，システムプログラミング実験の TA としてご協力頂きました東京大学 大学院情報理工学系研究科 五十嵐 将氏と五反田 正太郎氏に感謝申し上げます．

参考文献

[1] Wayne A. Christopher, Steven J. Procter, and Thomas E. Anderson. The Nachos Instructional Operating System. In *Proceedings of the USENIX Winter 1993 Conference*, pp. 481—489, 1993.

[2] David A. Holland, Ada T. Lim, and Margo I. Seltzer. A New Instructional Operating System. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, Vol. 34 of *ACM SIGCSE Bulletin*, pp. 111–115, 2002.

[3] Jason Nieh and Chris Vaill. Experiences Teaching Operating Systems Using Virtual Platforms and Linux. *ACM SIGOPS Operating Systems Review*, Vol. 40, No. 239, pp. 100–104, 2006.

[4] Rob Hess and Paul Paulson. Linux Kernel Projects for an Undergraduate Operating Systems Course. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, pp. 485–489, 2010.

[5] Adrian Kosowski, Michal Malafiejski, and Tomasz Noiński. Application of an Online Judge & Contester System in Academic Tuition. In *Proceedings of the 6th International Conference on Advances in Web Based Learning*, pp. 343–354, 2008.

[6] Mike Joy, Nathan Griffiths, and Russell Boyatt. The BOSS Online Submission and Assessment System. *Journal on Educational Resources in Computing*, Vol. 5, No. 3, pp. 2:1–2:28, 2005.

[7] Brenda Cheang, Andy Kurnia, Andrew Lim, and Wee-Chong Oon. On Automated Grading of Programming Assignments in an Academic Institution. *Computers and Education*, Vol. 41, No. 2, pp. 121–131, 2003.

[8] Miguel A. Revilla, Shahriar Manzoor, and Rujia Liu. Competitive Learning in Informatics: The UVaOnline Judge Experience. *Olympiads in Informatics*, Vol. 2, pp. 131–148, 2008.

[9] ICPC. `https://icpc.baylor.edu/`.

[10] 渡部有隆. オンラインジャッジの開発と運用-Aizu Online Judge-. 情報処理, Vol. 56, No. 10, pp. 998–1005, 2015.

[11] AtCoder. `https://atcoder.jp`.

[12] Dirk Merkel. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal*, Vol. 2014, , 2014.

[13] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, pp. 41–46, 2005.

[14] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. A Survey on Online Judge Systems and Their Applications. *ACM Computing Surveys*, Vol. 51, No. 1, pp. 3:1–3:34, 2018.

[15] DMOJ. `https://dmoj.ca/`.

[16] Lutece. `https://acm.uestc.edu.cn/home`.

[17] Jordi Petit, Omer Giménez, and Salvador Roura. Jutge.org: An Educational Programming Judge. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, pp. 445–450, 2012.

[18] David Hovemeyer, Jeffrey K. Hollingsworth, and Bobby Bhattacharjee. Running on the Bare Metal with GeekOS. *ACM SIGCSE Bulletin*, Vol. 36, pp. 315–319, 2004.

[19] Ben Pfaff, Anthony Romano, and Godmar Back. The Pintos Instructional Operating System Kernel. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, pp. 453–457, 2009.